Machine Learning

Master of Data Science

Alexandre Aussem

LIRIS UMR 5205 CNRS
Data Mining & Machine Learning Group (DM2L)
University of Lyon 1
Web: perso.univ-lyon1.fr/alexandre.aussem

October 31, 2024

Outline

Outline

Machine learning is a subfield of AI that involves using algorithms to enable machines to learn from data and make decisions.

- **Supervised Learning** : The algorithm is trained on labeled data, where the target output is known. The goal is to learn a mapping between input features and the corresponding targets. Ex. linear regression, logistic regression, decision trees, and support vector machines.
- **Unsupervised Learning**: The target values are missing. The goal may be to: Discover hidden patterns or structure in the data, known as clustering (e.g., k-means, hierarchical and spectral clustering), Determine the distribution of data within the input space, known as density estimation. project the data into a lower-dimensional space for the purpose of visualization or dimensionality reduction (e.g., PCA, t-SNE), and anomaly detection algorithms (e.g., outlier detection).
- **Semi-Supervised Learning**: Supervised techniques that also make use of a small amount of labeled data with a large amount of unlabeled data. The problem arises whenever the acquisition of labeled data requires a skilled human agent or a physical experiment. The cost associated with the labeling process is prohibitive, whereas acquisition of unlabeled data is relatively inexpensive. Ex. self-training algorithms (e.g., co-training), transductive learning algorithms, and graph-based algorithms (e.g., label propagation).

- **Reinforcement Learning**: The algorithm learns from interactions with an environment to make decisions that maximize a reward signal. The goal is to learn a policy that maps states to actions that lead to the highest reward over time. Ex. Q-learning, SARSA, and deep reinforcement learning algorithms (e.g., actor-critic methods, deep deterministic policy gradient).
- **Deep Learning**: Involves using neural networks with multiple layers to learn complex patterns in data. The algorithm learns by iteratively adjusting the weights of the network to minimize the difference between predicted and actual outputs. Ex. convolutional neural networks (CNNs), recurrent neural networks (RNNs, LSTM) networks, and transformers (BERT, GPT etc)
- **Transfer Learning**: A pre-trained model is fine-tuned on a new task to leverage the knowledge learned from the previous task. The goal is to improve performance on the new task while reducing the amount of training data required. Ex. using pre-trained word embeddings (e.g., Word2Vec, GloVe) for NLP tasks or pre-trained neural networks for image classification tasks.

- **Ensemble Learning**: Multiple models are combined to produce a better prediction than any single model. The goal is to leverage the strengths of each individual model and reduce the weaknesses. Examples include bagging (random forests), boosting (Gradient Boosting Machine), and stacking (team learning).
- **Time Series Analysis**: The algorithm learns patterns in sequential data over time to make predictions about future values. Examples include ARIMA models, recurrent nets and transformers.
- **Natural Language Processing (NLP)**: The algorithm learns to analyze and generate text data to perform tasks such as language translation, sentiment analysis, and text summarization. Examples include word embeddings, recurrent neural networks, and transformers.
- **Generative Adversarial Networks (GANs)**: Two algorithms compete with each other to generate new data that is indistinguishable from the original training data. The goal is to learn a generator network that can produce realistic data samples. Examples include image generation, video generation, and text generation.

Supervised learning : Classification vs Regression

A learning problem tries to predict properties of unknown data. In **supervised learning**, the data comes with additional attributes that we want to predict. We can separate supervised learning problems in a few large categories:

- **Classification**: Identifying to which category an object belongs to. Samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. Variants: *multi-class*, *multi-label classification*, *label ranking*, *collaborative filtering*.
- **Regression**: Predicting a continuous-valued attribute associated with an object. Variants: *multi-output learning* when the desired output consists of more continuous variables.
- **Common algorithms**: Decision Trees, ensemble methods (bagging, Random Forests, Boosting etc.) support vector machines (SVM), logistic regression, linear regression, naive Bayes classification, ridge regression, Lasso, k nearest neighbors, (deep) neural networks ...

Further problems in supervised learning

- **Structured (output) learning**: techniques that involve predicting structured objects, rather than scalar discrete or real values, e.g. translating a natural language sentence into a syntactic representation such as a parse tree, learning social networks, or gene regulatory networks, NLP, speech recognition, and computer vision. Bayesian networks and random fields, structured SVMs, Markov logic networks etc. Approximate inference and learning methods are usually needed.
- **Probabilistic graphical models**: large class of structured prediction models. Learn the statistical relationships between the variables. Training and prediction using a trained model are often computationally infeasible and approximate inference is used.
- **Time series classification**: Techniques that involve assigning time series patterns to specific categories or predicting future values based on past observations. The data is a *sequence* of discrete-time observations, not $i.i.d.$ anymore.
- **Collaborative filtering**: Recommender systems collect preferences or taste information from many users (collaborating) for making automatic predictions (filtering) about the interests of each user.

Further problems in supervised learning

- **Preprocessing**: Feature extraction and normalization.
- **Dimensionality reduction**: Reducing the number of random variables to consider for parsimony and increased efficiency : PCA, SVD, feature extraction and selection, non-negative matrix factorization, latent Dirichlet allocation (LDA) ...
- **Bias correction**; selection bias and omission bias.
- **Handling of missing values**.
- **Model selection**: Comparing, validating and choosing parameters and models for improved accuracy via parameter tuning. Grid search, cross validation ...

Choosing the right estimator

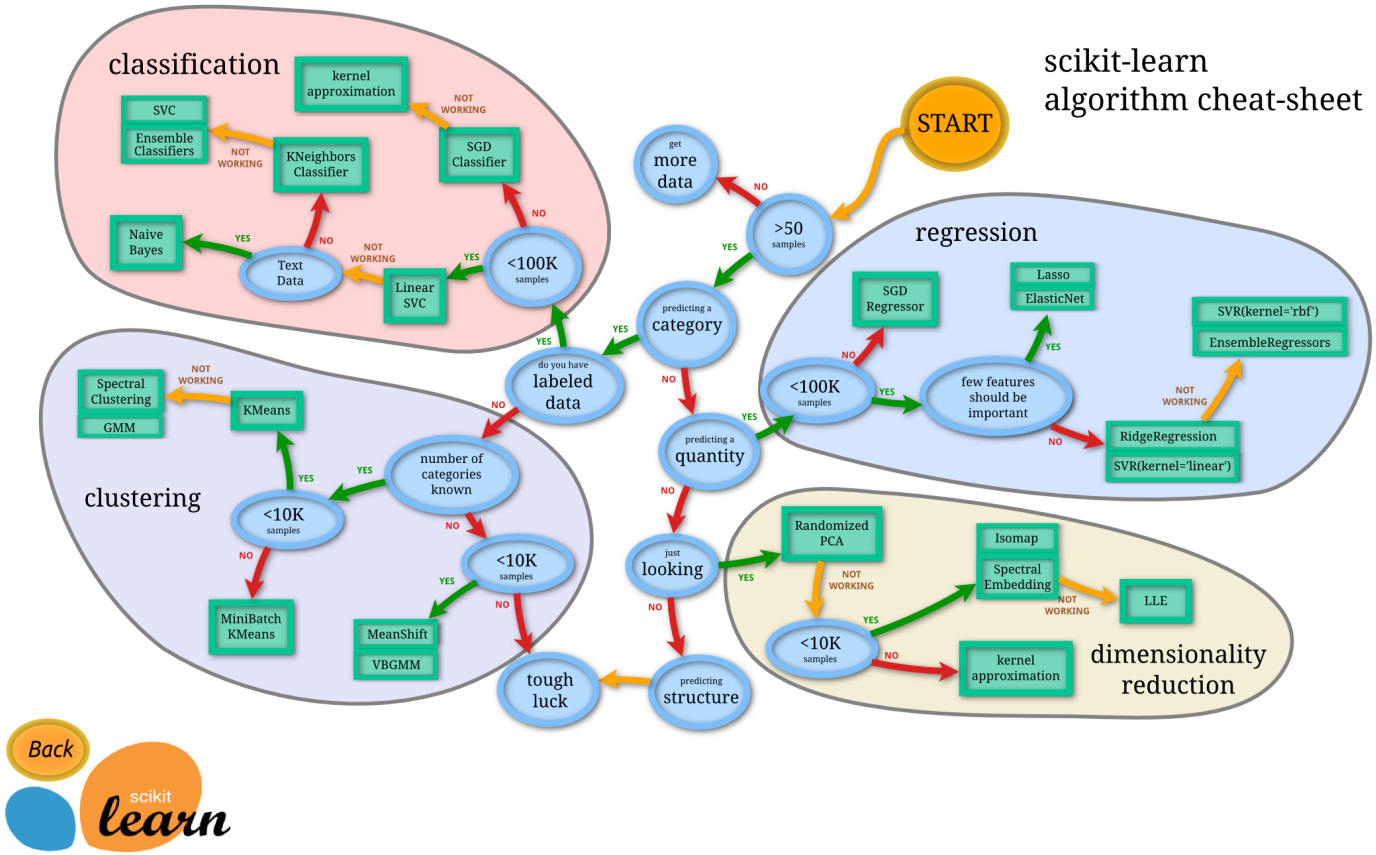


Figure: A road map on how to approach problems with regard to which estimators to try on your data.

Model selection: illustration

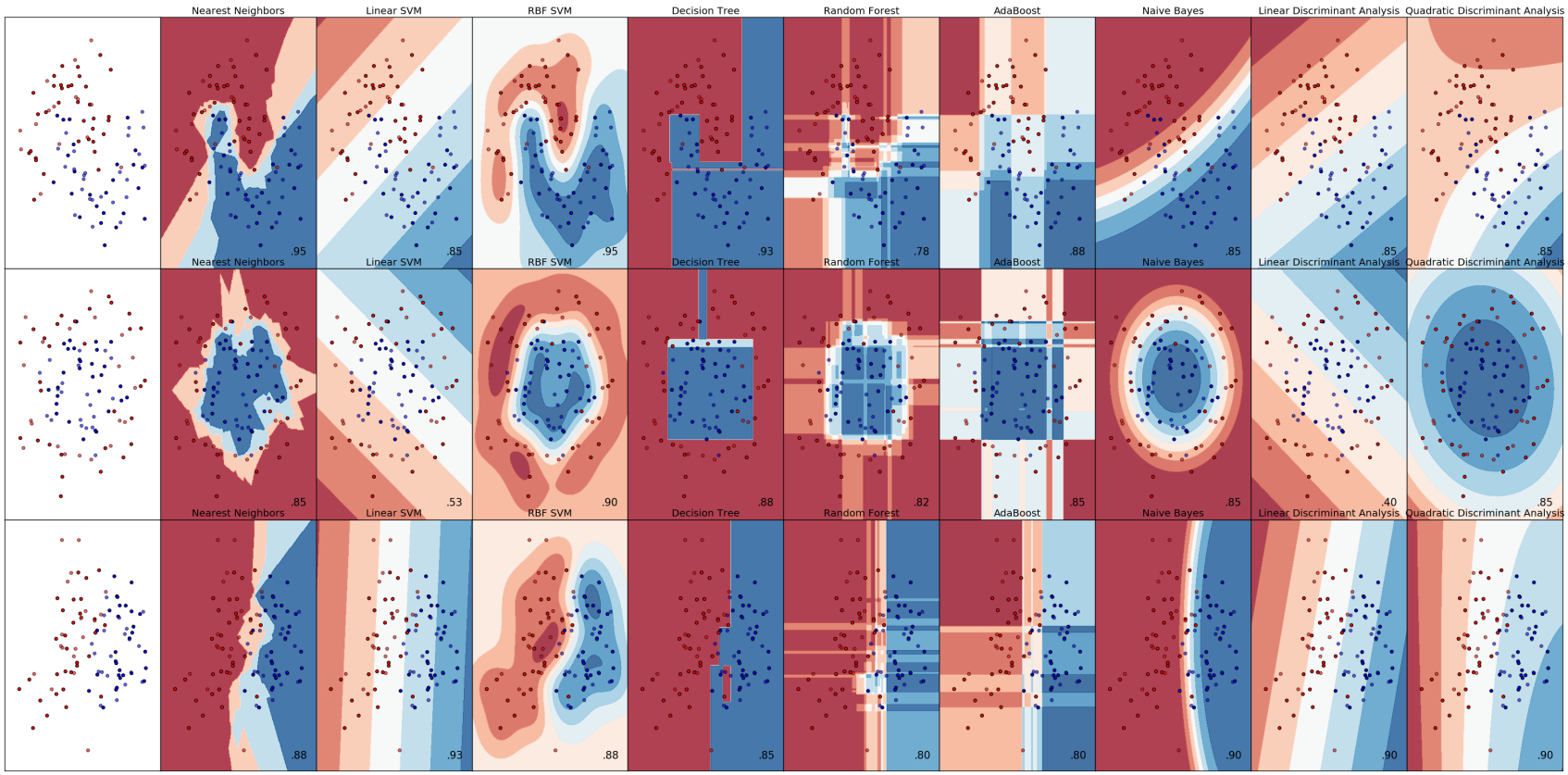


Figure: A comparison of several classifiers in scikit-learn on synthetic bivariate data.

Model comparisons: illustration

Approach	ACC		AUC		RMS	
	L is tuned	L=200	L is tuned	L=200	L is tuned	L=200
Ad	0.846±0.11	0.857±0.10	0.880±0.13	0.893±0.12	0.676±0.13	0.668±0.10
AdET	0.784±0.13	**0.862±0.09**	0.796±0.14	**0.898±0.12**	0.572±0.13	**0.667±0.09**
AdSt	0.847±0.12	0.833±0.11	0.898±0.12	0.874±0.13	0.603±0.11	0.598±0.08
ArcX4	0.866±0.09	0.852±0.09	**0.920±0.09**	0.892±0.11	**0.715±0.10**	0.686±0.10
ArcX4ET	0.866±0.09	0.868±0.08	**0.921±0.09**	0.901±0.10	0.715±0.10	0.693±0.09
Bag	**0.858±0.08**	0.823±0.10	**0.914±0.10**	0.875±0.12	**0.714±0.09**	0.660±0.10
BagET	**0.865±0.10**	0.836±0.11	**0.916±0.11**	0.893±0.11	**0.717±0.10**	0.673±0.10
Logb	0.848±0.10	0.845±0.10	0.880±0.12	0.884±0.13	**0.679±0.14**	0.635±0.09
RF	0.865±0.09	0.864±0.09	**0.915±0.11**	0.896±0.12	**0.716±0.10**	0.689±0.10
RadP	0.859±0.08	0.850±0.09	**0.915±0.09**	0.889±0.13	**0.714±0.09**	0.669±0.09
RadPET	0.864±0.10	0.861±0.09	0.915±0.11	0.908±0.10	**0.716±0.10**	0.680±0.09
Rot	0.864±0.09	0.865±0.08	0.916±0.10	0.903±0.11	**0.722±0.10**	0.700±0.10
Rotb	0.862±0.09	0.865±0.09	0.913±0.10	0.897±0.11	0.719±0.10	0.702±0.11
RotbET	0.864±0.09	0.866±0.09	0.913±0.10	0.900±0.11	0.719±0.10	0.704±0.11
RotET	0.864±0.09	0.871±0.08	0.915±0.10	0.901±0.10	**0.721±0.10**	0.698±0.10
Swt	0.851±0.10	0.859±0.09	0.899±0.10	0.888±0.11	**0.692±0.08**	0.638±0.07
SwtET	0.864±0.10	0.866±0.08	0.913±0.11	0.890±0.11	**0.699±0.09**	0.649±0.08
Vad	0.812±0.10	**0.858±0.09**	0.817±0.13	**0.894±0.12**	0.628±0.14	**0.684±0.11**
VadET	0.791±0.13	**0.864±0.08**	0.792±0.15	**0.899±0.11**	0.601±0.15	**0.681±0.09**

Terminology

Common terms, synonyms or closely related topics:

- Machine Learning
- Statistical Learning
- Computational Learning Theory
- Knowledge Discovery in Databases
- Pattern Recognition
- Data Mining
- Artificial Intelligence

$\neq$ Cognitive science ...

Illustration: Handwritten digit recognition

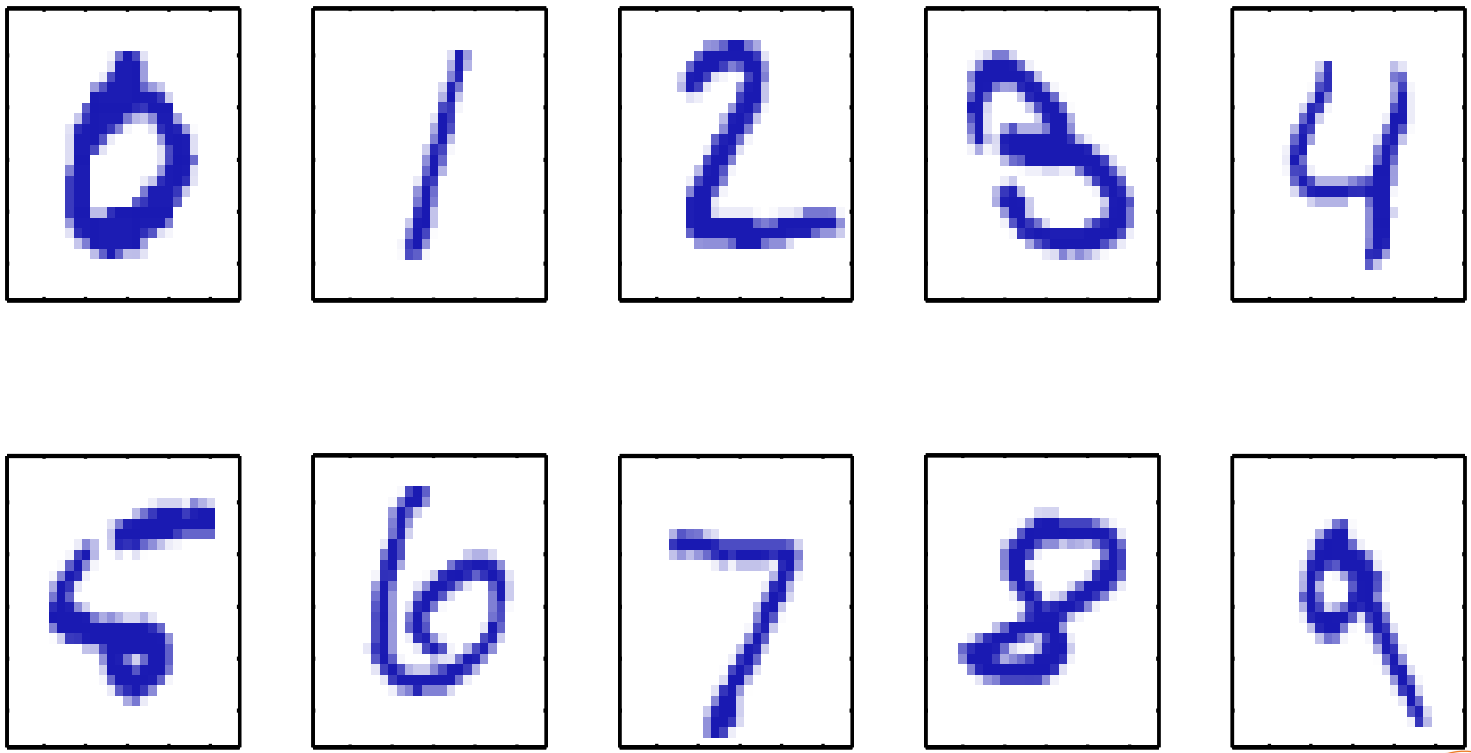


Illustration: Handwritten digit recognition

- Each digit corresponds to a 28×28 pixel image, represented by a vector comprising 784 real numbers. The goal is to build a machine that will take such a vector x as input and that will produce the identity of the digit $0, \ldots, 9$ as the output.
- This is a nontrivial problem due to the **wide variability of handwriting**.
- The original input variables are typically preprocessed (feature extraction) to transform them into some new space of variables
- The precise form of the function $f(x)$ is determined during the training phase,

Image annotation/retrieval

Target 1:	cloud	yes/no
Target 2:	sky	yes/no
Target 3:	tree	yes/no
...	...	...

Multi-label learning

- Training data: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, $\boldsymbol{y}_i \in \mathcal{Y} = \{0, 1\}^m$
- **Predict** the vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ for a given $\boldsymbol{x}$.

	X_1	X_2	Y_1	Y_2	$\ldots$	Y_m
$\boldsymbol{x}_1$	5.0	4.5	1	1		0
$\boldsymbol{x}_2$	2.0	2.5	0	1		0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_n$	3.0	3.5	0	1		1
$\boldsymbol{x}$	4.0	2.5	?	?		?

Multivariate prediction

- Training data: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, $\boldsymbol{y}_i \in \mathcal{Y} = \mathbb{R}^m$
- **Predict** the vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ for a given $\boldsymbol{x}$.

	X_1	X_2	Y_1	Y_2	$\ldots$	Y_m
$\boldsymbol{x}_1$	5.0	4.5	14	0.3		9
$\boldsymbol{x}_2$	2.0	2.5	15	1.1		4.5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_n$	3.0	3.5	19	0.9		2
$\boldsymbol{x}$	4.0	2.5	?	?		?

Label ranking

- Training data: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$, where $\boldsymbol{y}_i$ is a ranking (permutation) of a fixed number of labels/alternatives.
- **Predict** permutation $(y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(m)})$ for a given $\boldsymbol{x}$.

	X_1	X_2	 Y_1	 Y_2	 Y_m
$\boldsymbol{x}_1$	5.0	4.5	1	3	2
$\boldsymbol{x}_2$	2.0	2.5	2	1	3
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_n$	3.0	3.5	3	1	2
$\boldsymbol{x}$	4.0	2.5	?	?	?

Multi-task learning

- Training data: $\{(\boldsymbol{x}_{1j}, y_{1j}), (\boldsymbol{x}_{2j}, y_{2j}), \ldots, (\boldsymbol{x}_{nj}, y_{nj})\}$, $j = 1, \ldots, m$, $y_{ij} \in \mathcal{Y} = \mathbb{R}$.
- **Predict** y_j for a given $\boldsymbol{x}_j$.

	X_1	X_2	Y_1	Y_2	$\ldots$	Y_m
$\boldsymbol{x}_1$	5.0	4.5	14			9
$\boldsymbol{x}_2$	2.0	2.5		1.1		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{x}_n$	3.0	3.5				2
$\boldsymbol{x}$	4.0	2.5				?

Collaborative filtering

- Training data: $\{(u_i, m_j, y_{ij})\}$, for some $i = 1, \ldots, n$ and $j = 1, \ldots, m$, $y_{ij} \in \mathcal{Y} = \mathbb{R}$.
- **Predict** y_{ij} for a given u_i and m_j.

	m_1	m_2	m_3	$\cdots$	m_m
u_1	1			$\cdots$	4
u_2	3		1	$\cdots$	
u_3		2	5	$\cdots$	
$\ldots$				$\cdots$	
u_n		2		$\cdots$	1

Dyadic prediction

			4	5	⋯	7	8	6
			10	14	⋯	9	21	12
instances			y_1	y_2	⋯	y_m	y_{m+1}	y_{m+2}
1	1	x_1	10	?	⋯	1	?	?
3	5	x_2		0.1	⋯	0		?
7	0	x_3	?	?	⋯	1	?	
1	1	...			⋯	0		?
3	1	x_n		0.9	⋯	1	?	?
2	3	x_{n+1}	?		⋯	?		?
3	1	x_{n+2}		?	⋯	?	?	?

Further problems

- Structured output prediction
- Multi-task learning and transfer learning
- Matrix factorization
- Sequence learning, time series prediction and data stream mining
- Metric learning
- Topic Modelling
- Causal inference
- ...

Softwares

Name	Advantages	Disadvantages	Open Source
R	Library support and Visualization	Steep learning curve	Yes
Matlab	Native matrix support, Visualization	Expensive, incomplete statistics support	No
Scientific Python	Ease and Simplicity	Heavy development	Yes
Excel	Easy, Visual, Flexible	Large datasets	No
SAS	Large Datasets	Expensive, outdated programming language	No
Stata	Easy Statistical Analysis		No
SPSS	Like Stata but more expensive and less flexible		

Supervised learning

- In the setting of supervised learning, a function of $f(., \mathbf{w}) : X \to Y$ is to be learned, that predicts well on instances that are drawn from a **joint probability distribution** $p(\mathbf{x}, y)$ on $X \times Y$.
- The true distribution $p(\mathbf{x}, y)$ is **unknown**. Instead, one has access to a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$.
- In statistical learning models, the training sample $(\mathbf{x}_i, y_i)$ are assumed to have been drawn i.i.d. from $p(\mathbf{x}, y)$.

Generalization error

- Given a loss function $L(f(\mathbf{x}, \mathbf{w}), y)$ $X \times X \rightarrow \mathbb{R}$ that measures the difference between the predicted value $f(\mathbf{x}, \mathbf{w})$ and the true value y. The objective is to minimize the expected "risk":

True loss, generalization error or **expected risk** :

$$E(\mathbf{w}) = \mathbb{E}[L(f(\mathbf{X}, \mathbf{w}), Y)] = \int \int L(f(\mathbf{x}, \mathbf{w}), y) p(\mathbf{x}, y) d\mathbf{x} dy$$

- The True loss is expressed as an expectation over the unknown **joint probability distribution** $p(\mathbf{x}, y)$.

Supervised learning

- The ideal goal is to select a function $f(., \mathbf{w}) \in \mathcal{H}$, where $\mathcal{H}$ is a space of functions called a **hypothesis space**, so that the true loss is minimised.
- As $p(\mathbf{x}, y)$ is unknown, a common paradigm is to estimate a function $\hat{f}$ through (regularized) **empirical risk minimization**.
- The loss function may be chosen for numerical reasons (e.g. convexity).
- Distinct loss functions may lead to functions $\hat{f}$ that differ significantly in their predictions.

Standard loss functions

- In **regression**, $L(\cdot)$ is typically the **squared loss**:

$$\mathbb{E}[L(f(X,\mathbf{w}),Y)] = \int\int (f(\mathbf{x},\mathbf{w})-y)^2 p(\mathbf{x},y)d\mathbf{x}dy$$

- In **classification**, $L(\cdot)$ is typically the **zero-one loss**:

$$\begin{aligned}\mathbb{E}[L(f(X,\mathbf{w}),Y)] &= P_{(X,Y)}[f(X,\mathbf{w}) \neq Y] \\ &= \sum_y \int \mathbb{I}[f(\mathbf{x},\mathbf{w}) \neq y] p(\mathbf{x},y)d\mathbf{x}\end{aligned}$$

- In **probabilstic classification**, $f((X,\mathbf{w}),y)$ is an estimate for $P(y|\mathbf{x})$ and $L(\cdot)$ is typically the **logarithmic loss**:

$$\mathbb{E}[L(f(X,\mathbf{w}),Y)] = -\int\int \log\left(f((x,\mathbf{w}),y)\right) p(\mathbf{x},y)d\mathbf{x}dy$$

MAP estimate in classification

- In classification, $L(f(\boldsymbol{x}), y) = \mathbb{I}(f(\mathbf{x}) \neq y)$ called the **0-1 loss**. ($\mathbf{w}$ is omitted for conciseness). It is convenient to denote by C_k the class k. So

$$\arg\min_{f \in \mathcal{H}} \sum_{y} \int \mathbb{I}[f(\mathbf{x}, \mathbf{w}) \neq y] p(\mathbf{x}, y) dy$$

$$= \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{K} \int \mathbb{I}(f(\mathbf{x}) \neq C_k) p(\mathbf{x}, C_k) d\boldsymbol{x}$$

$$= \arg\min_{f \in \mathcal{H}} \int \left\{ \sum_{i=1}^{K} \mathbb{I}(f(\mathbf{x}) \neq C_k) p(C_k | \boldsymbol{x}) \right\} p(\boldsymbol{x}) d\boldsymbol{x}$$

- For all $\boldsymbol{x}$, the optimal solution $f(\boldsymbol{x})$ satisfies

$$\arg\min_{f \in \mathcal{H}} \left\{ \sum_{i=1}^{K} \mathbb{I}(f(\mathbf{x}) \neq C_k) p(C_k | \boldsymbol{x}) \right\}$$

MAP estimate in classification

- When $L(f(\boldsymbol{x}), y)$ is **0-1 loss**, we can prove that MAP estimate minimizes 0-1 loss,

$$\begin{aligned}\arg\min_{f\in\mathcal{H}} \sum_{i=1}^{K} L[f(\mathbf{x}), C_k]P(C_k|\mathbf{x}) &= \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{K} \mathbb{I}(f(\mathbf{x}) \neq C_k)P(C_k|\mathbf{x}) \\ &= \arg\min_{f\in\mathcal{H}} [1 - P(f(\mathbf{x}) = C_k|\mathbf{x})] \\ &= \arg\max_{f\in\mathcal{H}} P(f(\mathbf{x}) = C_k|\mathbf{x})\end{aligned}$$

- The optimal solution for the 0-1 loss is the most likely class (the mode of the distribution).

Posterior mean minimizes quadratic loss

- For continuous outputs, a more appropriate loss function is **squared error** or **quadratic loss**.
- The posterior expected loss is given by

$$\int_y L[f(\mathbf{x}), y] p(y|\mathbf{x}) \mathrm{d}y = \int_y [y - f(\mathbf{x})]^2 p(y|\mathbf{x}) \mathrm{d}y$$
$$= \int_y \left[y^2 - 2yf(\mathbf{x}) + f(\mathbf{x})^2 \right] p(y|\mathbf{x}) \mathrm{d}y$$

Posterior mean minimizes quadratic loss

- Setting the derivative w.r.t. f equal to zero gives

$$\frac{\partial \rho}{\partial f} = \int_y [-2y + 2f(\mathbf{x})]p(y|\mathbf{x})\mathrm{d}y = 0 \Rightarrow$$

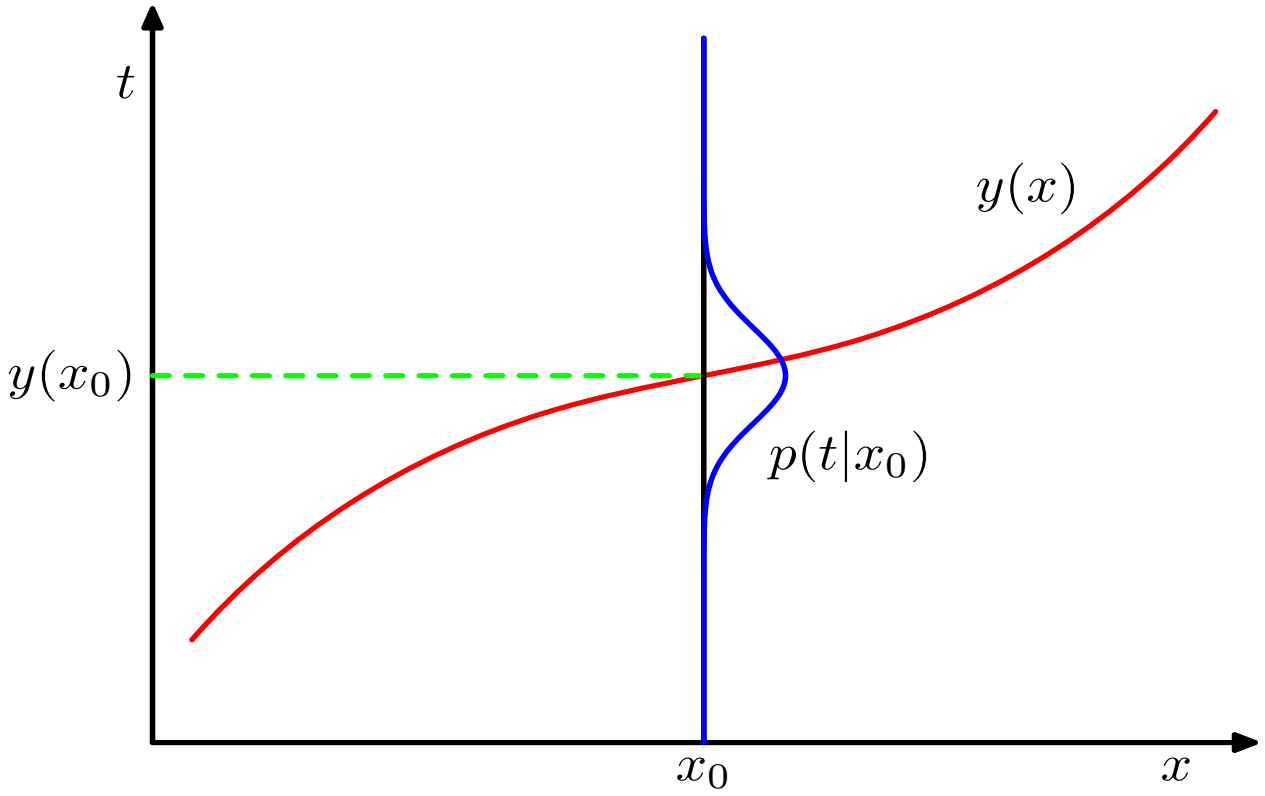
$$\int_y f(\mathbf{x})p(y|\mathbf{x})\mathrm{d}y = \int_y yp(y|\mathbf{x})\mathrm{d}y$$

$$f(\mathbf{x}) \int_y p(y|\mathbf{x})\mathrm{d}y = \mathbb{E}_{p(y|\mathbf{x})}[Y] = \mathbb{E}[Y|\mathbf{x}]$$

$$\Rightarrow f(\mathbf{x}) = \mathbb{E}[Y|\mathbf{x}]$$

- Hence the the **minimum mean squared error** solution or **MMSE** is the **posterior mean**.

Squared loss



The regression function $y(\mathbf{x})$, which minimizes the expected squared loss, is given by the mean of the conditional distribution $p(t|x)$.

Loss functions

- There are situations in which squared loss can lead to very poor results in regression and where more sophisticated approaches are needed. An important example concerns situations in which the conditional distribution $p(y|\mathbf{x})$ is multimodal.
- A simple generalization of the squared loss, called the *Minkowski* loss, is given by

$$\mathbb{E}[L(f(\mathbf{X}, \mathbf{w}), Y)] = \int \int |f(\mathbf{x}, \mathbf{w}) - y|^q p(\mathbf{x}, y) d\mathbf{x} dy$$

- Is there a closed-form expression for the optimal solution $f^\star(\cdot)$?

Loss functions

$$\mathbb{E}[L(f(\mathbf{X}, \mathbf{w}), Y)] = \int \int |f(\mathbf{x}, \mathbf{w}) - y|^q p(\mathbf{x}, y) d\mathbf{x} dy$$

One can show that the optimal solution, $\hat{f}(\mathbf{x})$, reduces to

- The conditional **mean**, $\mathbb{E}[Y|\mathbf{x}]$, for $q = 2$ (i.e., quadratic loss function)
- The conditional **median** of $p(y|\mathbf{x})$ for $q = 1$ (i.e., absolute loss function)
- The conditional **mode** of $p(y|\mathbf{x})$ for $q \to 0$ (i.e., zero-one loss function)

Is there $\mathbf{w}^\star$ such that $f(\cdot, \mathbf{w}^\star) = \hat{f}(\cdot)$? Otherwise the model will be biased.

Posterior median minimizes the absolute loss

- A more robust alternative to the quadratic loss is the absolute loss. The optimal estimate is the **posterior median**,

$$\rho = \int_y L[f(\mathbf{x}), y] p(y|\mathbf{x}) \mathrm{d}y = \int_y |y - f(\mathbf{x})| p(y|\mathbf{x}) \mathrm{d}y$$

$$= \int_{y<f(\mathbf{x})} (f(\mathbf{x}) - y) p(y)|\mathbf{x}) \mathrm{d}y + \int_{y \geq f(\mathbf{x})} (y - f(\mathbf{x})) p(y|\mathbf{x}) \mathrm{d}y$$

$$\frac{\partial \rho}{\partial f} = \int_{y<f(\mathbf{x})} p(y|\mathbf{x}) \mathrm{d}y - \int_{y \geq f(\mathbf{x})} p(y|\mathbf{x}) \mathrm{d}y = 0$$

$$\Rightarrow P(Y < f(\mathbf{x})|\mathbf{x}) = P(Y \geq f(\mathbf{x})|\mathbf{x}) = 0.5$$

Loss function and consistency

- The loss function should approximate the actual cost we are paying due to misclassifcation errors.
- There are many loss functions that are **non-convex and discontinuous**. It is difficult to optimize these losses directly (e.g. gradient-based optimization).
- The loss functions used for training classifiers (e.g. exponential loss, hinge loss) are usually approximations or **surrogates** of the zero-one-loss (misclassification rate).
- In practice, we consider a surrogate loss function which can be optimized by efficient algorithms.
- A learning algorithm is **consistent** if the expected risk of a learned function converges to the Bayes risk as the training sample size increases.

More complex loss functions with multiple labels

- So far, each instance was associated with a single label Y. In real-world applications, one object is usually relevant to multiple labels $\{Y_1, \ldots, Y_L\}$.
- Ranking these labels is a central part of many **information retrieval** problems, such as document retrieval, collaborative filtering, sentiment analysis, and online advertising.
- The **rank-loss** evaluates the label order induced by giving a numerical score for each item in some sense:

$$L_{rankloss}(f(\mathbf{x}), \mathbf{y}) = \frac{1}{n^+ \cdot n^-} \sum_{y_i \leq y_j} \mathbb{I}[f_i(\boldsymbol{x}) \geq f_j(\boldsymbol{x})]$$

  where $n^+ = \{y_i : y_i = +1\}$ and $n^- = \{y_j : y_j = -1\}$.
- It is difficult to find surrogate loss consistent with the ranking loss.

Empirical loss

- The true loss $E(\mathbf{w})$ can not be computed exactly as the distribution $p(\mathbf{x}, y)$ is **unkown**. We may estimate the expected risk from $\mathcal{D}$ using the approximation,

$$\mathbb{E}[f(\mathbf{X})] = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \simeq \frac{1}{n}\sum_{j=1}^{n} f(\mathbf{x}_j)$$

- So, the intent is to find a function $f(., \mathbf{w})$ that minimizes the empirical loss,

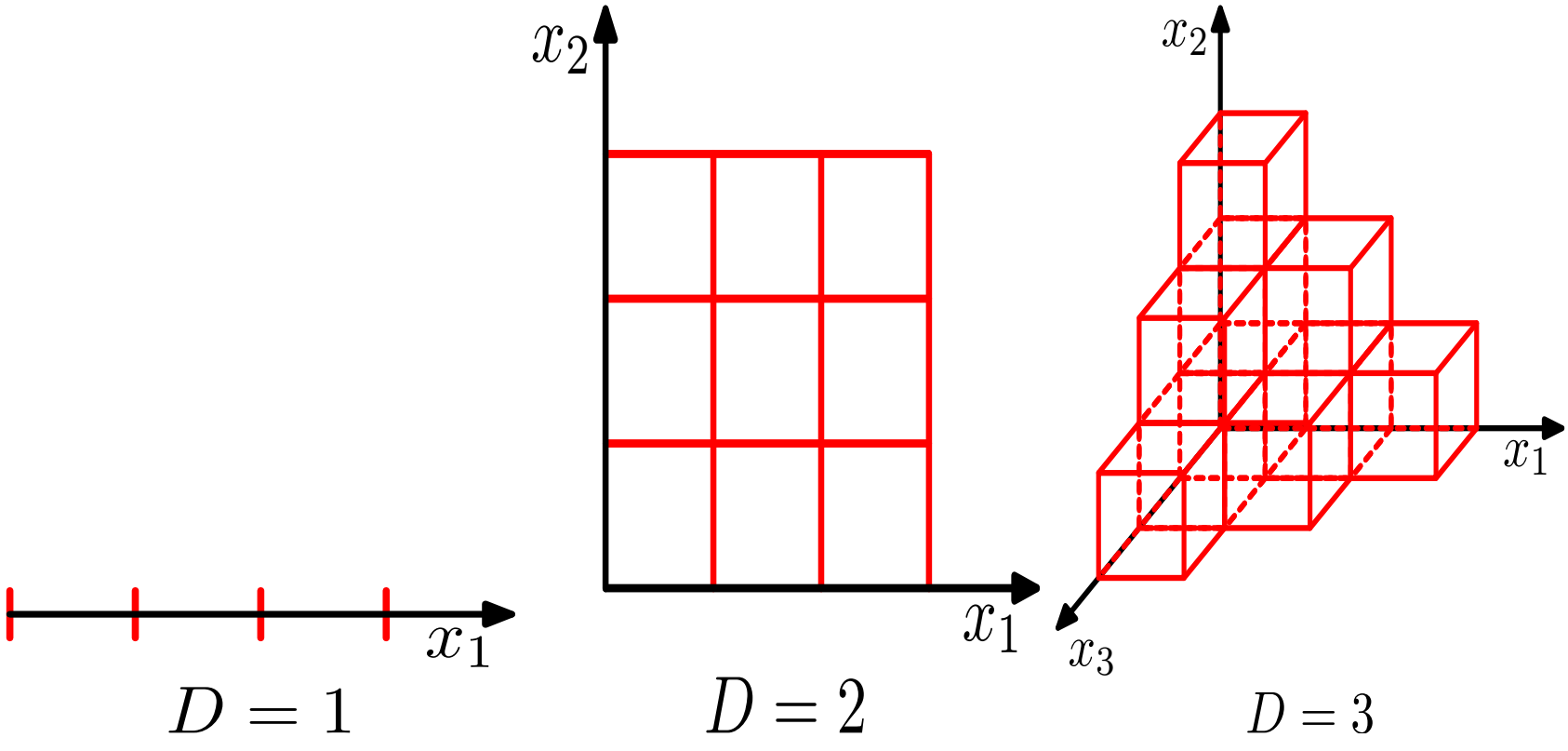
Empirical loss:

$$\hat{E}(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n} L(f(\mathbf{x}_j, \mathbf{w}), y_j)$$

The curse of dimensionality

- In practical applications, we have to deal with spaces of high dimensionality comprising many input variables
- Suppose the input space is divided into cells and any new test point is assigned to the class that has a majority number of representatives in the same cell as the test point.

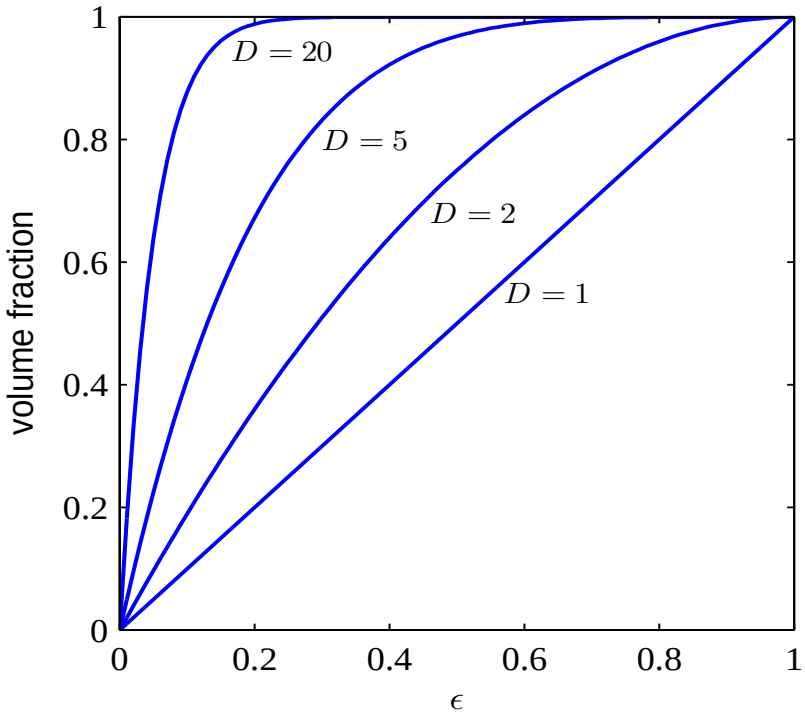
The curse of dimensionality

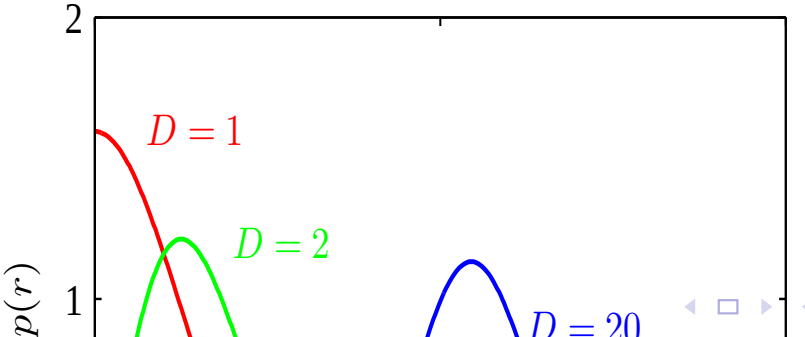


The curse of dimensionality

- As the number of such cells grows exponentially with D, we need an exponentially large quantity of training data in order to ensure that the cells are not empty
- For a polynomial of order M, the number of coefficients to estimate varies as $O(D^M)$. More **parsimonious** models are needed.
- Our geometrical intuitions fail badly when we consider spaces of higher dimensionality. Learning algorithms based on the euclidean metric suffer drastically from the curse of dimensionality.

The curse of dimensionality





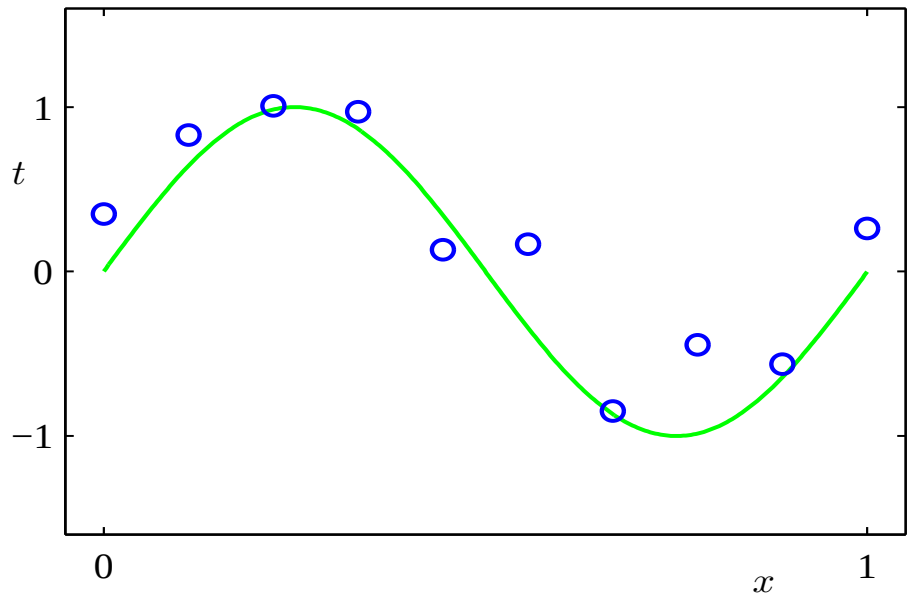
The curse of dimensionality

- With high-dimensional data, the **Euclidean distance is not informative** because all vectors are almost equidistant to the search query vector.
- Local methods work best in low-dimensional embeddings.
- Example: The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression (majority vote of its k neighbors or average of their values) where the function is only approximated locally.
- Dimension reduction is usually performed prior to applying the k-NN algorithm to high-dimensional data ($d > 10$).

Binary classification

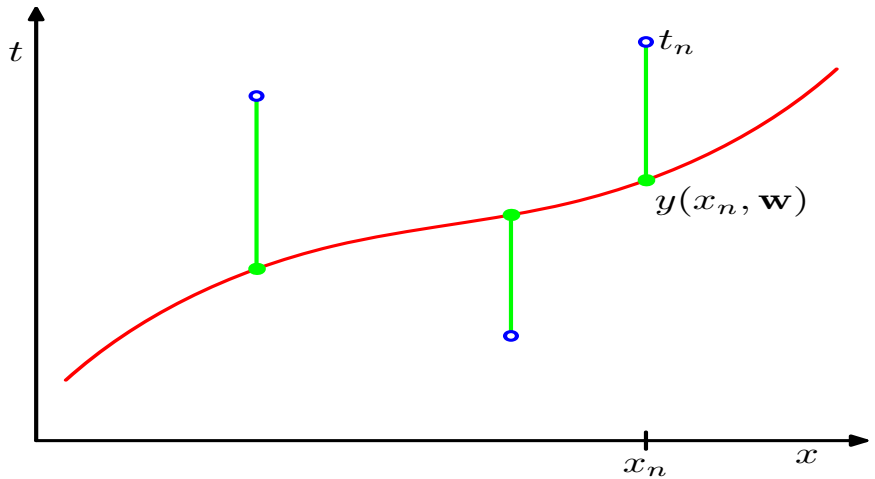
- Many ML problems can be cast as **binary classification problems**.
- The goal is to infer a function $f : X \to \{-1, +1\}$ from $\mathcal{D}$ such that the expected risk, $P_{(X,Y)}[f(X, \mathbf{w}) \neq y]$, is as low as possible.
- The model output value is often regarded as **score** of the positive class that is compared to some arbitrary **threshold** θ.
- A **probabilistic** classifier output an estimate of $P(Y = 1|\mathbf{x})$, but **calibration** of the probabilities is often required when the loss is expressed as an expected cost.

Example: Polynomial curve fitting



$$f(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

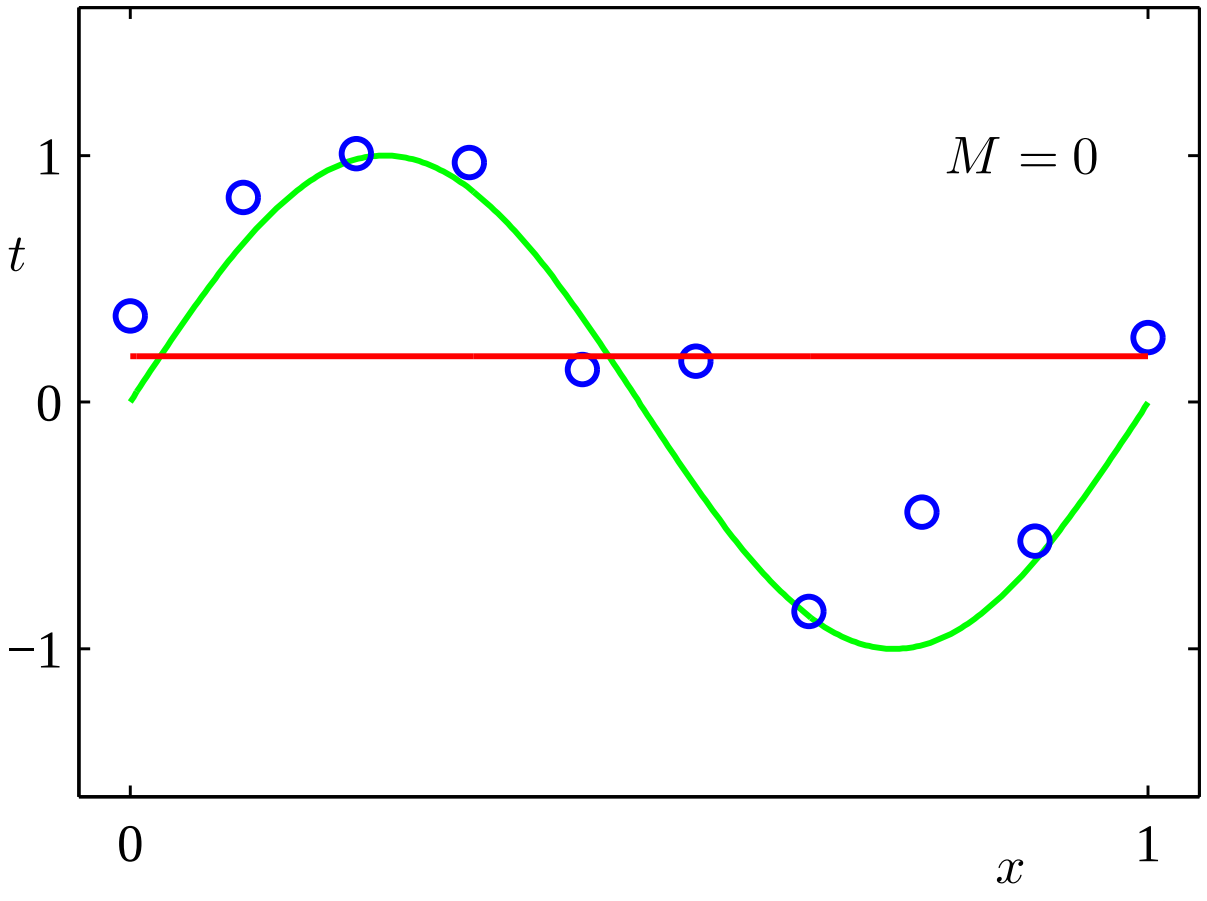
Sum of the squares



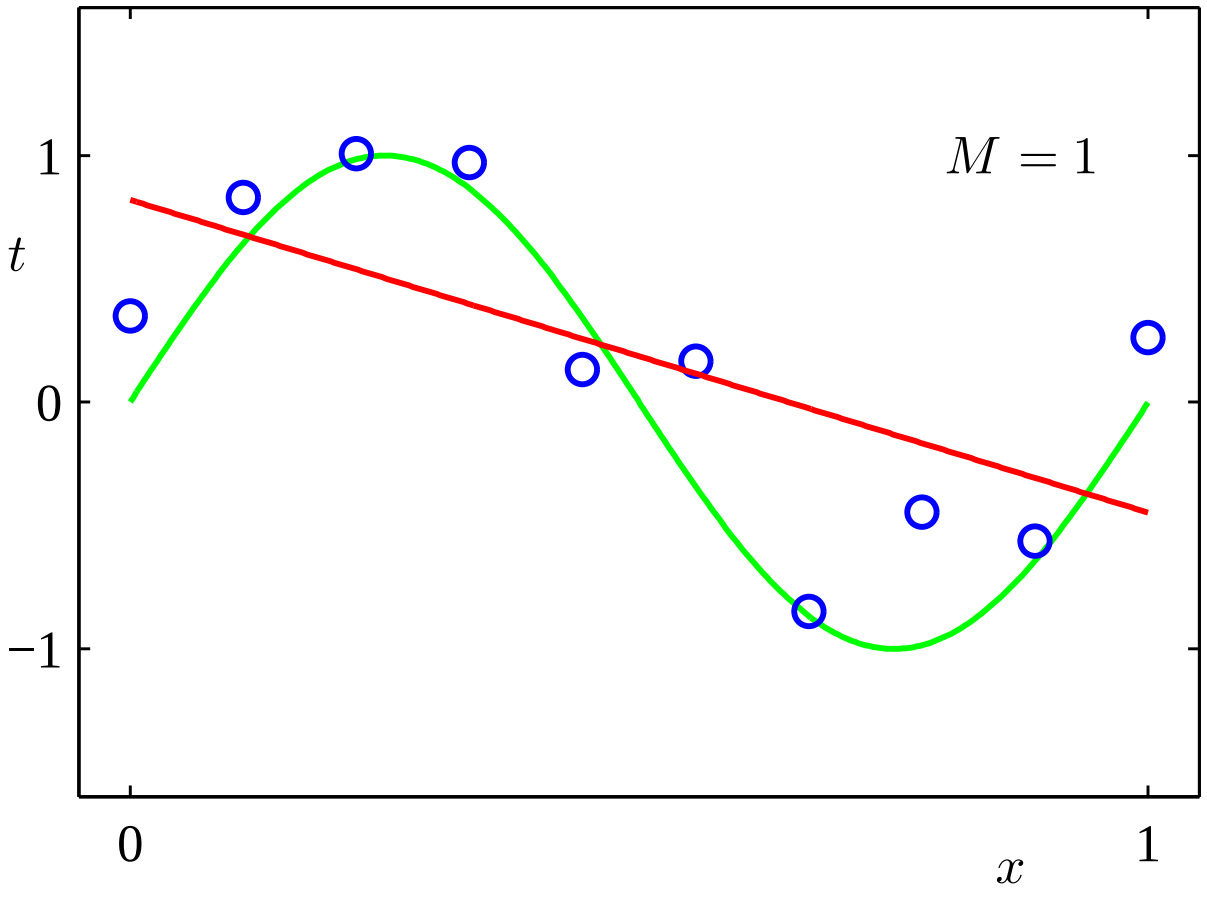
The sum of the squares of the errors measures the misfit between the function $y(x, \mathbf{w})$, for any given value of $\mathbf{w}$, and the training set data points,

$$\hat{E}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \{f(x_n, \mathbf{w}) - y_n\}^2$$

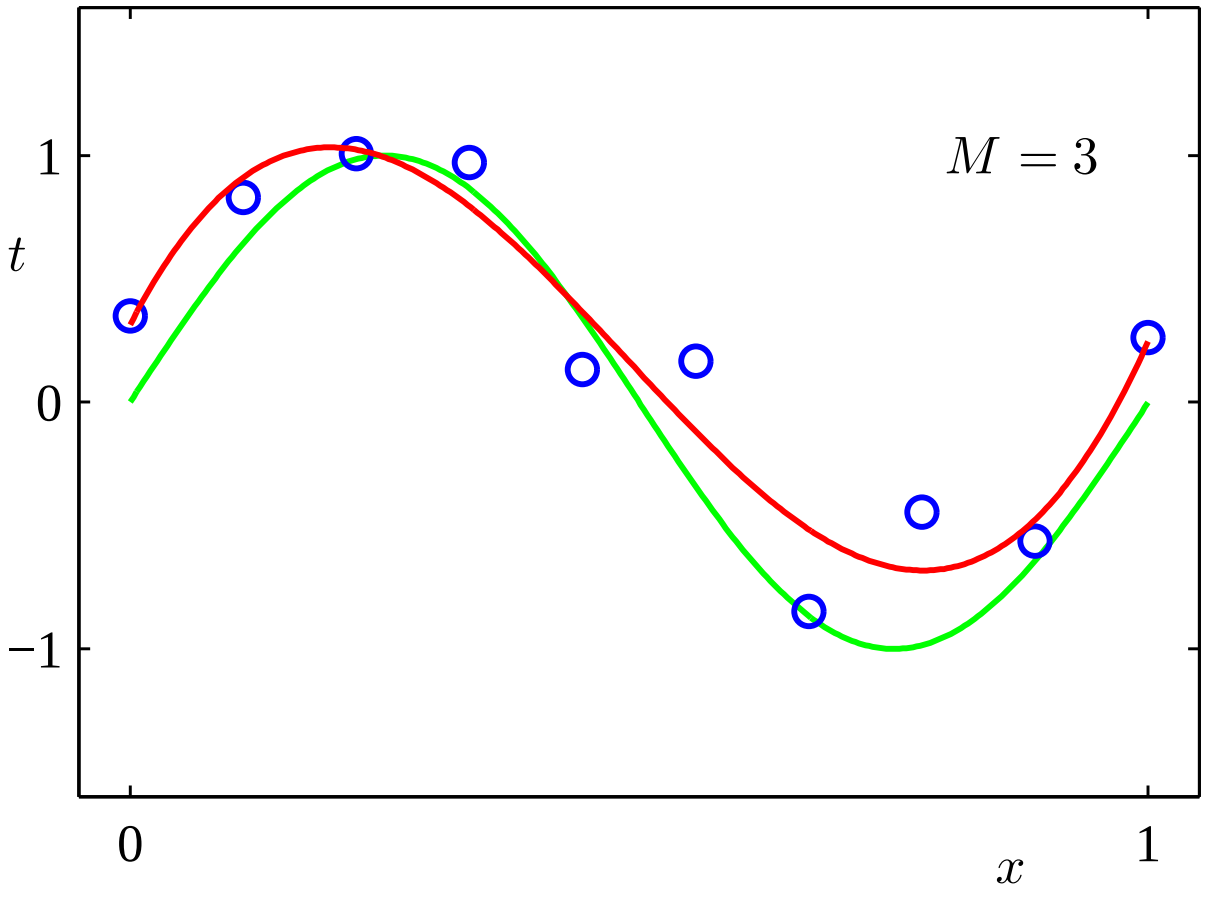
Polynomial of order 0



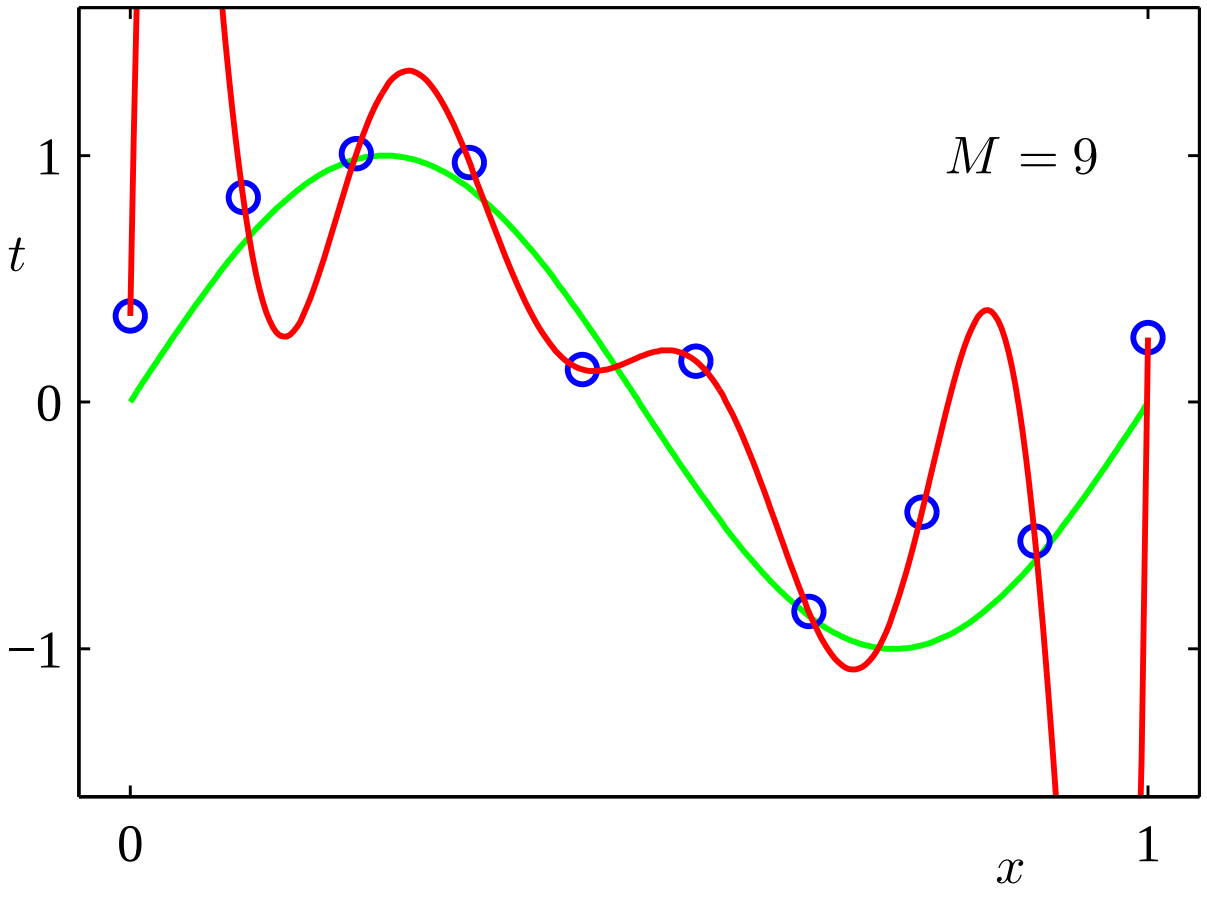
Polynomial of order 1



Polynomial of order 3



Polynomial of order 9

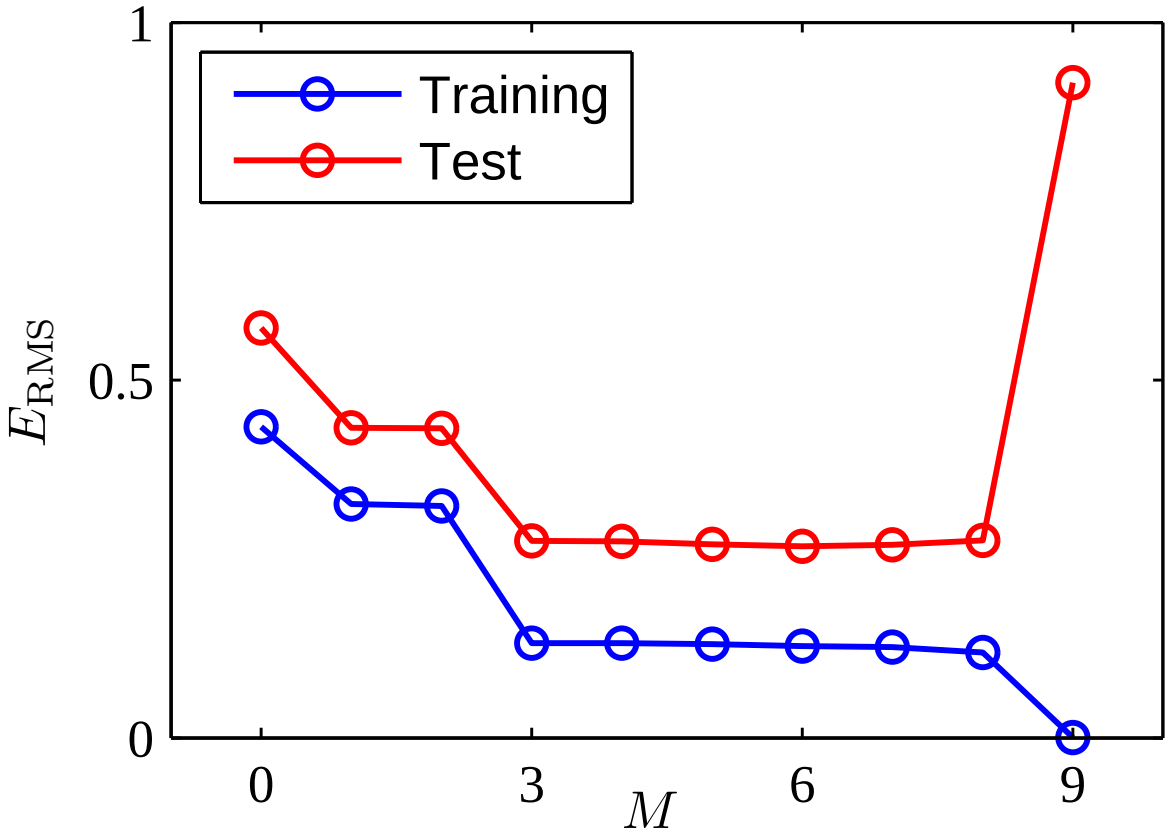


Polynomial of order 9

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

We observe that, as M increases, the magnitude of the coefficients typically gets larger, i.e. $|w_j| \to \infty$ when $M \to \infty$.

Over-fitting



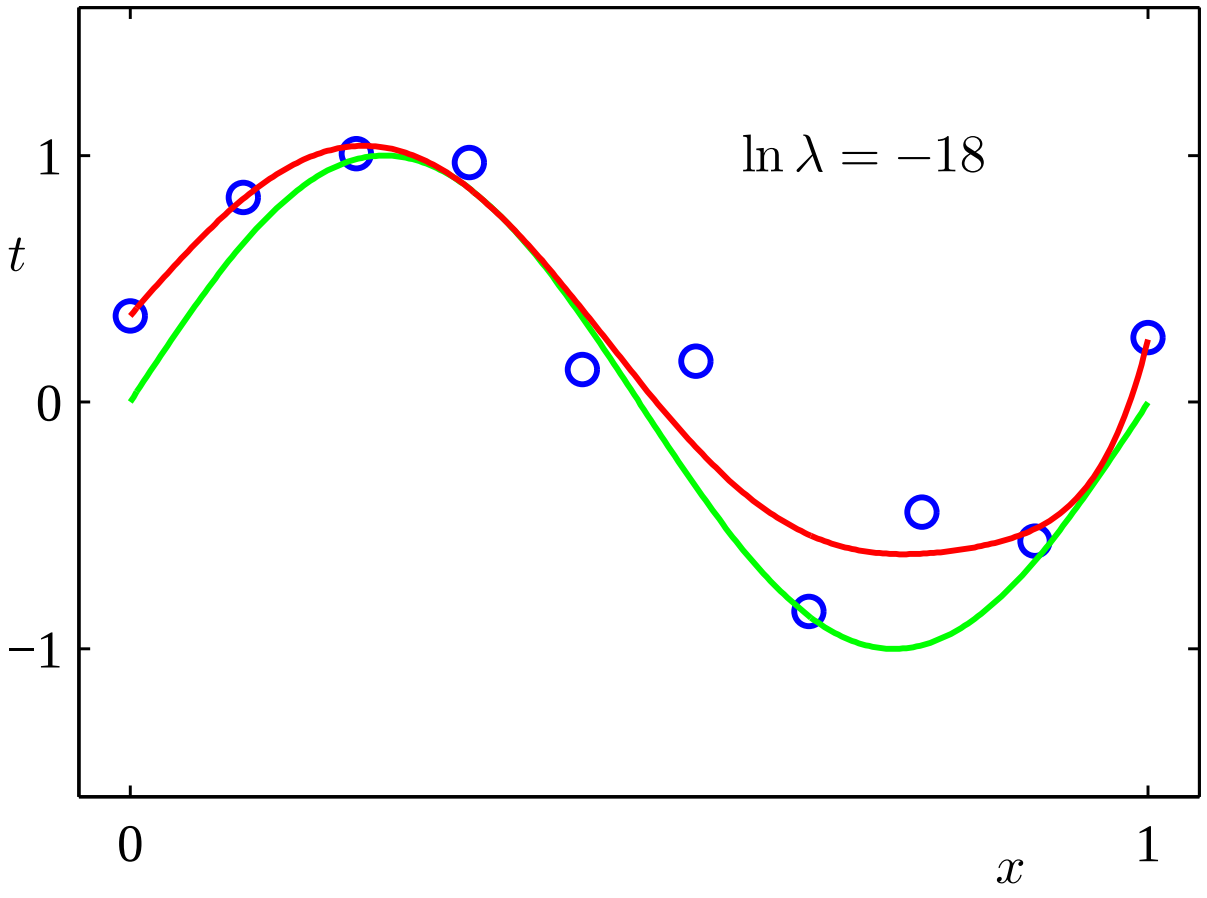
Regularization

- Regularization is one technique often used to control the over-fitting phenomenon. It involves adding a penalty term to the error function to discourage the coefficients from reaching large values,

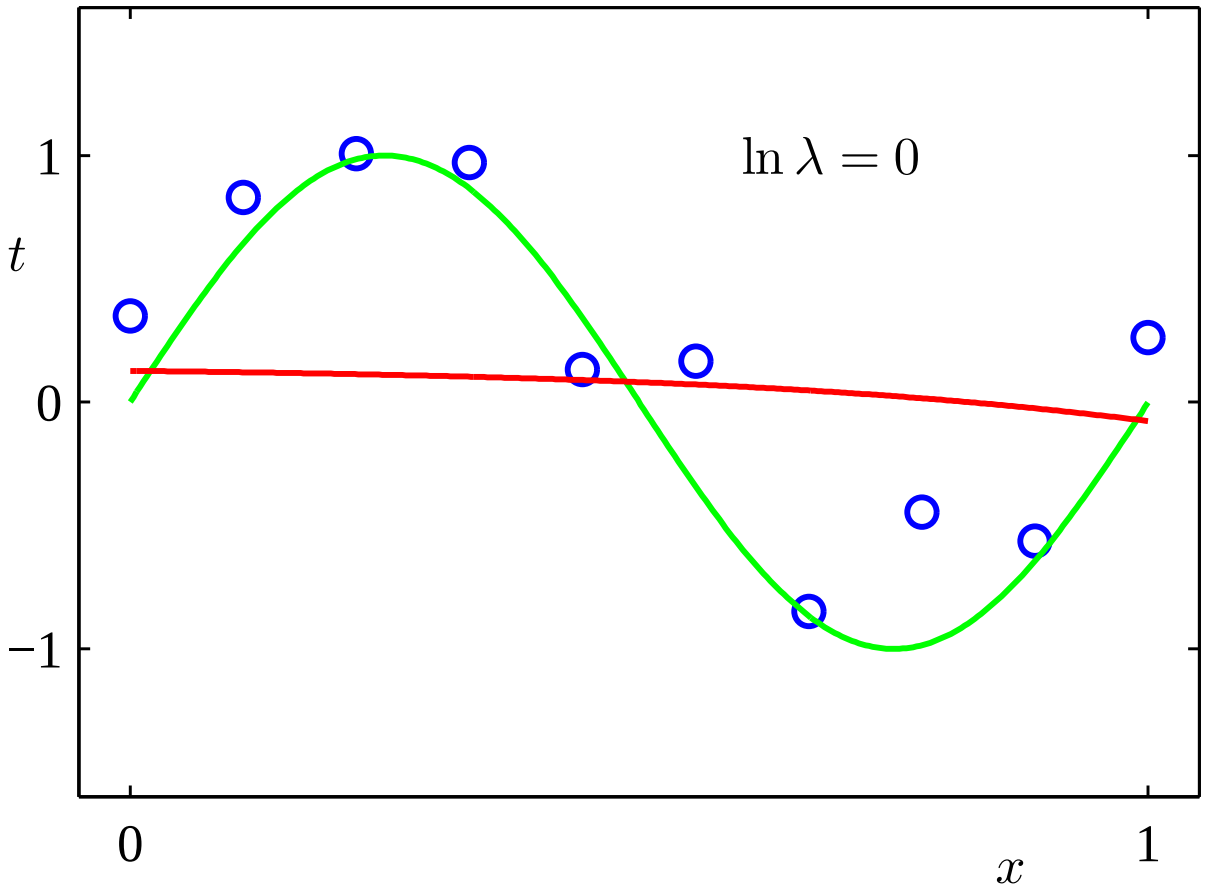
$$\hat{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{M}\{f(x_n, \mathbf{w}) - y_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- λ controls the effective complexity of the model and hence determines the degree of over-fitting.

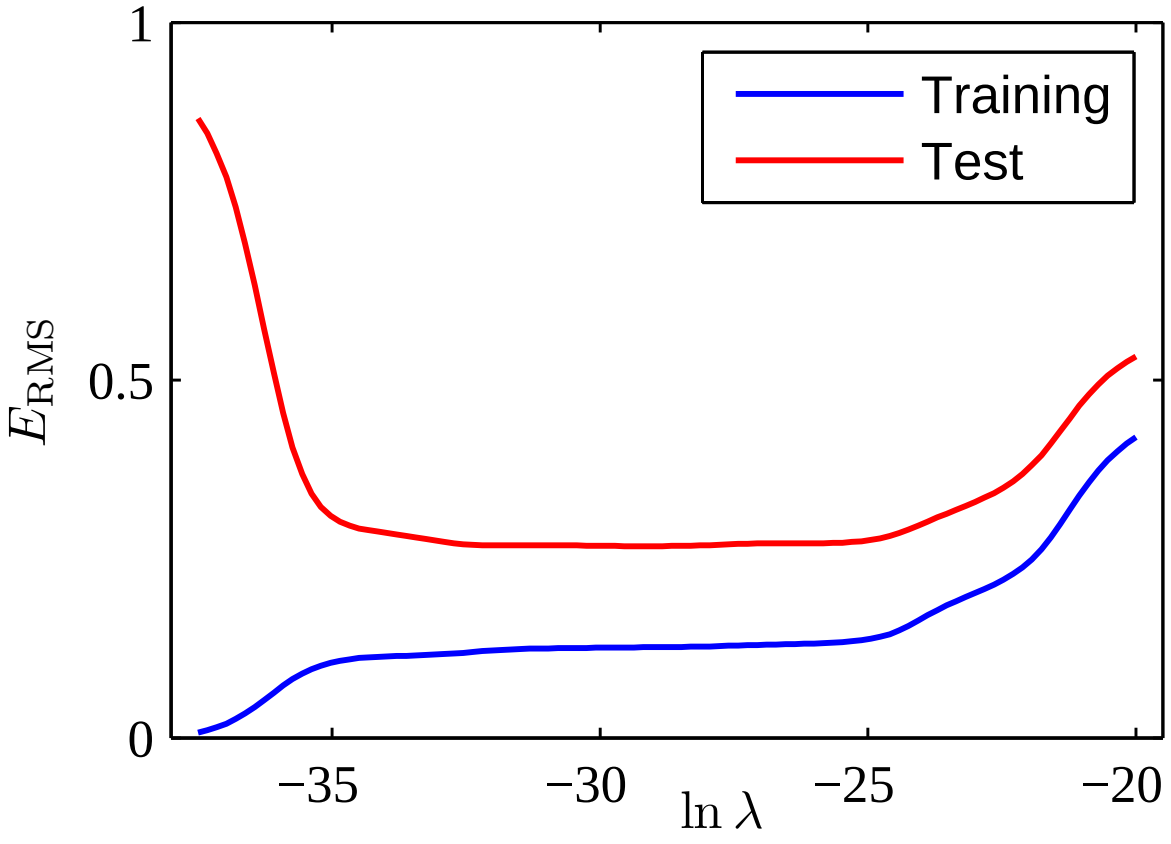
Regularization $\ln \lambda = -18$



Regularization $\ln \lambda = 0$



Regularization : E_{RMLS} vs. $\ln \lambda$



Polynomial of order 9

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

We see that, as λ increases, the magnitude of the coefficients typically gets larger.

Bayes optimal prediction

- In regression, we typically assume that $y = f(x) + \epsilon$ where ϵ is a white noise with variance σ^2. the expected squared loss can be written in the form:

$$\begin{aligned}
\mathbb{E}[\mathbf{w}] &= \int\int (f(x,\mathbf{w}) - y)^2 f_{X,Y}(x,y)dxdy \\
&= \int\int (f(x,\mathbf{w}) - \mathbb{E}[Y|x])^2 f_{X,Y}(x,y)dxdy \\
&+ \int \{\int (\mathbb{E}[Y|x] - y)^2 f_{Y/X}(y)dy\} f_X(x)dx
\end{aligned}$$

  where the conditional expectation $\mathbb{E}[Y|x] = \int yp(y|x))dy$, is the **Bayes optimal prediction** for the squared loss function,

- The last term, equal to σ^2, is the **intrinsic noise** on the data and represents the minimum achievable value of the expected loss.

The Bias-Variance decomposition

- $f(x, \mathbf{w})$ is dependent of $\mathcal{D}$. Taking the expectation of this expression with respect to $\mathcal{D}$,

$$\begin{aligned} \mathbb{E}_{\mathcal{D}}[(f(x,\mathbf{w}) - \mathbb{E}[Y|x])^2] &= (\mathbb{E}_{\mathcal{D}}[(f(x,\mathbf{w})] - \mathbb{E}[Y|x])^2 \\ &+ \mathbb{E}_{\mathcal{D}}[(f(x,\mathbf{w}) - \mathbb{E}_{\mathcal{D}}[(f(x,\mathbf{w})])^2] \end{aligned}$$

we get the following decomposition of the expected squared loss:

$$\textbf{Error} = \textbf{(bias)}^2 + \textbf{variance} + \textbf{noise}$$

- The model with the optimal predictive capability is the one that leads to the best balance between bias and variance.

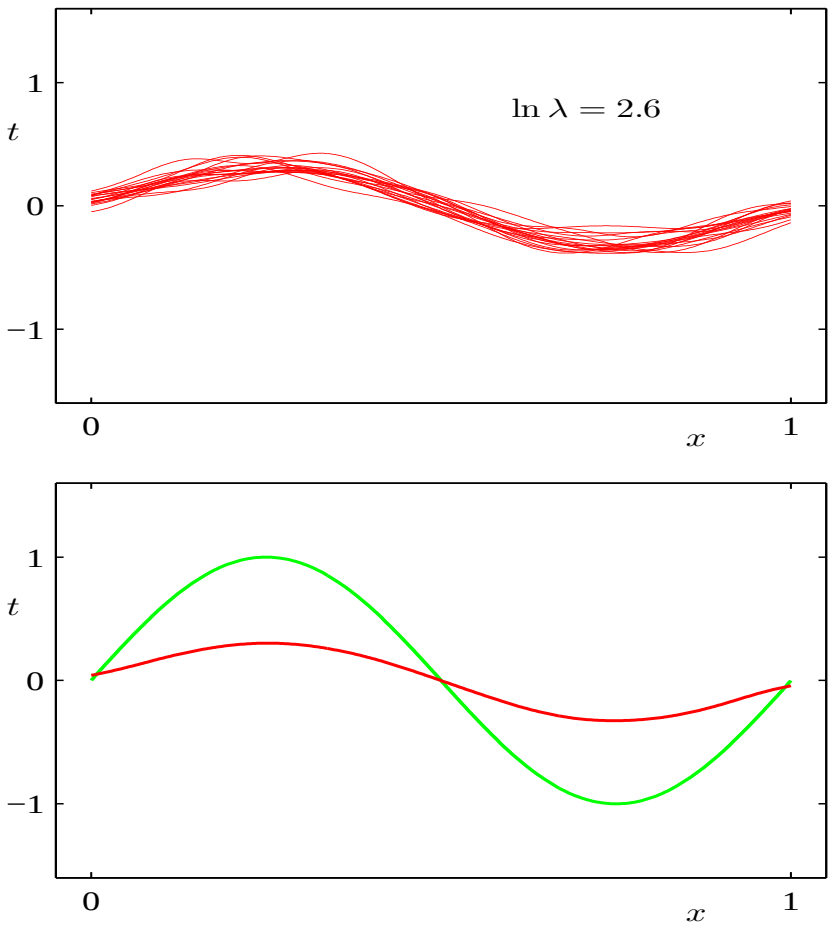
Regularization

- Regularization is one technique often used to control the over-fitting phenomenon. It involves adding a penalty term to the error function to discourage the coefficients from reaching large values,

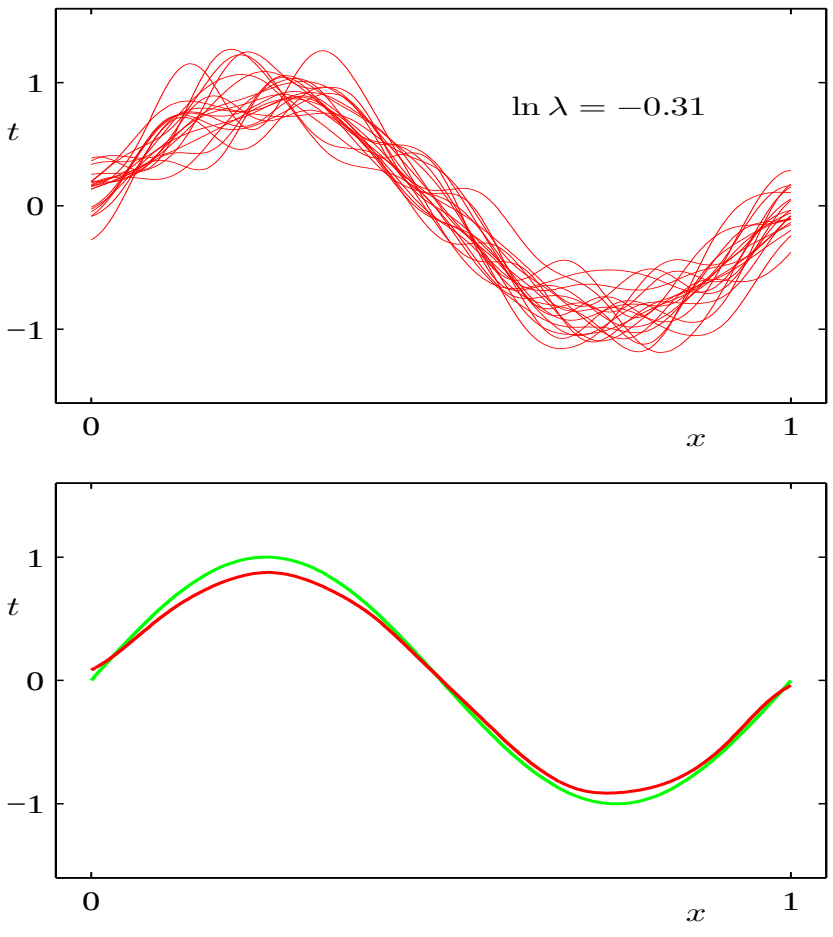
$$\hat{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{M}\{f(x_n, \mathbf{w}) - y_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- λ **controls the bias-variance trade-off**. Bias represents the extent to which the average prediction over all data sets differs from the desired regression function. Variance measures the extent to which the solutions for individual data sets vary around their average

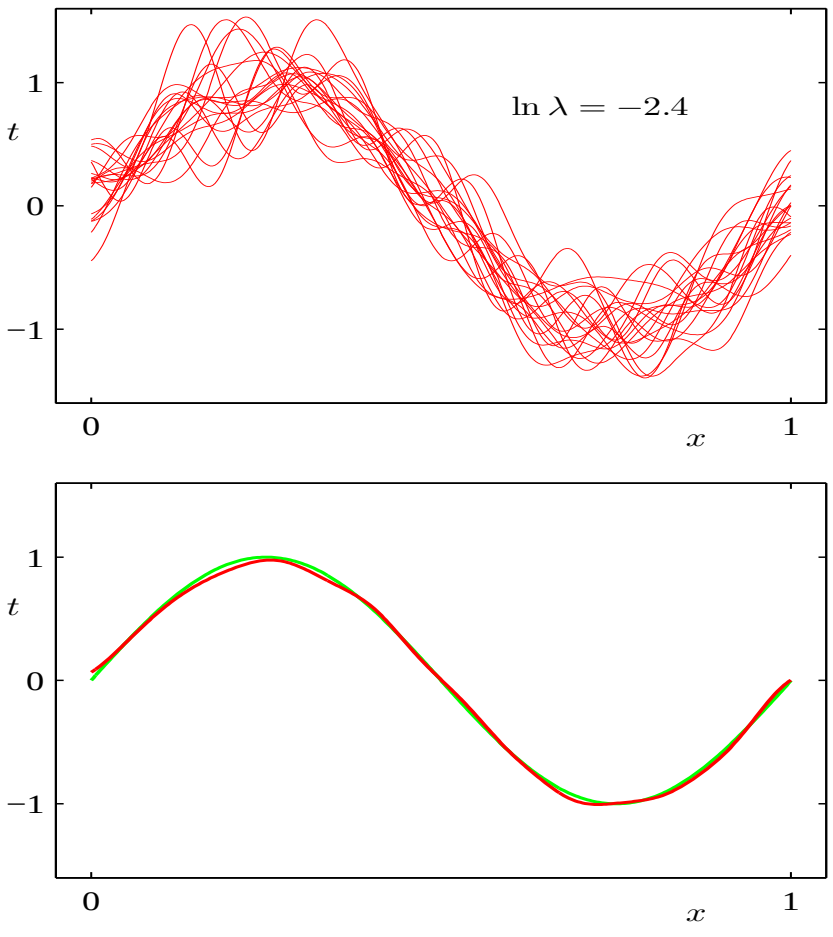
The bias-variance decomposition as a fonction of ln λ



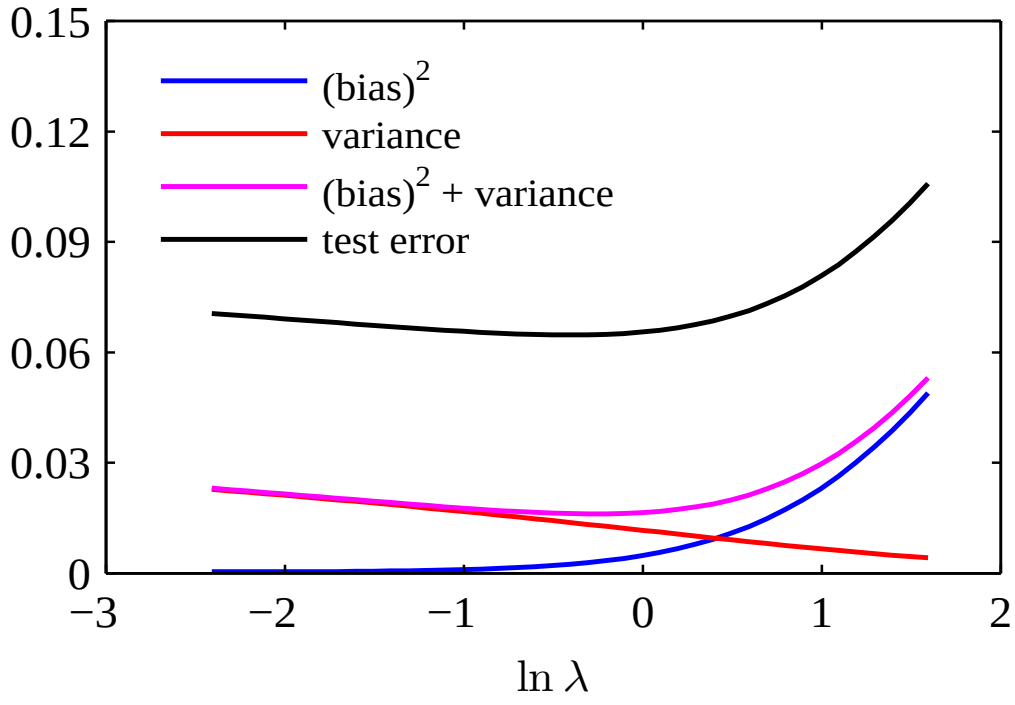
The bias-variance decomposition as a fonction of ln λ



The bias-variance decomposition as a fonction of ln λ



The bias-variance decomposition



Very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance.

A bias-variance decomposition in classification

Several bias-variance decompositions in classification exist, one is given by

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[P(Y \neq f(X, \mathbf{w}))] &= 1 - P(j^\star|X) \\ &+ \mathbb{E}_X[(P(j^\star|X) - P(\hat{j}|X))P(\hat{j}|f, X)] \\ &+ \mathbb{E}_X[\sum_{j \neq \hat{j}}((P(j^\star|X) - P(j|X))P(j|f, X))]\end{aligned}$$

with notations

$$\begin{aligned}P(j|f, x)) &= P_{\mathcal{D}}(f(x, \mathbf{w}) = j|X = x)) \\ P(j|x)) &= P(Y = j|x)) \\ j^\star(x) &= \underset{j}{\text{argmax}} P(j|x) \\ \hat{j}(x) &= \underset{j}{\text{argmax}} P(j|f, x)\end{aligned}$$

The bias-variance decomposition

We obtain the following decomposition of the expected squared loss

Bias-variance decomposition in regression

$$\textbf{Error} = \textbf{(bias)}^2 + \textbf{variance} + \textbf{noise}$$

Bias-variance decomposition in classification

$$\textbf{Error} = \textbf{bias} + \textbf{spread} + \textbf{Bayes error}$$

The *spread* acts as variance en regression. Noise and Bayes error are irreducible. The goal is to find the best balance between bias and variance (or spread).

Decision Theory

- Probability theory provides a consistent mathematical framework for quantifying and manipulating uncertainty.
- Determination of $p(x, t)$ from a set of training data is typically a difficult inference problem in high dimensional spaces (curse of dimensionality).
- We also need a decision theory that, when combined with probability theory, allows us to make **optimal decisions in situations involving uncertainty**.
- We need a rule that assigns each value of x to one of the available classes given by $p(x, t)$. These decision should be optimal in some appropriate sense (**cost dependent**).

Decision Theory

- We need a rule that that divide the input space into regions $\mathcal{R}_k$ called **decision regions**, one for each class, such that all points in $\mathcal{R}_k$ are assigned to class $\mathcal{C}_k$.
- The boundaries between decision regions are called **decision boundaries** or decision surfaces.

Decision Theory

- Suppose $p(\mathbf{x}, y)$ is given with $y \in \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$, we are interested in $p(\mathcal{C}_k|\mathbf{x})$. Using Bayes' theorem,

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

- We can interpret $p(\mathcal{C}_k)$ as the prior probability for the class $\mathcal{C}_k$, and $p(\mathcal{C}_k|\mathbf{x})$ as the corresponding posterior probability.
- If our aim is to minimize the chance of assigning x to the wrong class, we choose the **class having the higher posterior probability**.
- This amounts to choose k s.t. $p(\mathcal{C}_k|\mathbf{x}) > p(\mathcal{C}_j|\mathbf{x})$, $\forall j \neq k$. In this case $\mathcal{R}_k = \{\mathbf{x}|p(\mathcal{C}_k|\mathbf{x}) > p(\mathcal{C}_j|\mathbf{x}), \forall j \neq k\}$.

Minimizing the misclassification rate

- Note that

$$p(\mathcal{C}_k|\mathbf{x}) > p(\mathcal{C}_j|\mathbf{x})$$

  is equivalent to

$$p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k) > p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)$$

- So, instead of learning $p(\mathcal{C}_k|\mathbf{x})$, we may learn K distinct models $p(\mathbf{x}|\mathcal{C}_k)$ for $k = 1, \ldots, L$. The $p(\mathcal{C}_k)$ are easily estimated..

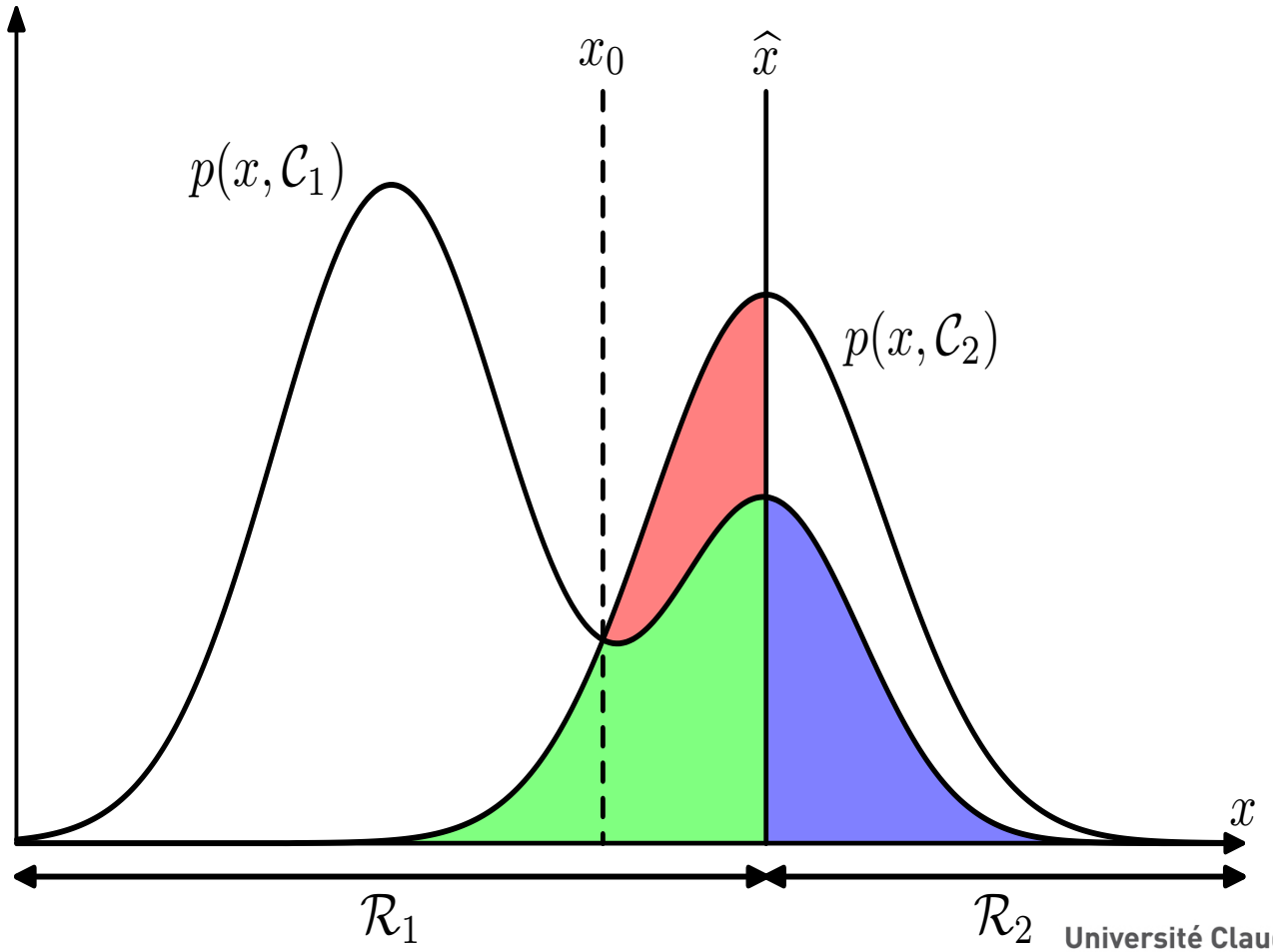
Minimizing the misclassification rate

- In the 2-class problem, the probability an error occurring is given by

$$\begin{aligned} P(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) + p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) \\ &= \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) dx + \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) dx \end{aligned}$$

- Clearly to minimize $P(\text{mistake})$ we should arrange that each x is assigned to whichever class has the smaller value.

Minimizing the misclassification rate



Minimizing the expected loss

- The misclassification rate is not appropriate when the consequences of the mistakes can be dramatically different, we need an overall measure of loss (or cost function) incurred in taking any of the available decisions.
- By assigning a new value of x, with true class $\mathcal{C}_k$, to class $\mathcal{C}_j$ (where j may not be equal to k), we incur some level of loss that we denote by L_{kj}.
- The average loss is given by

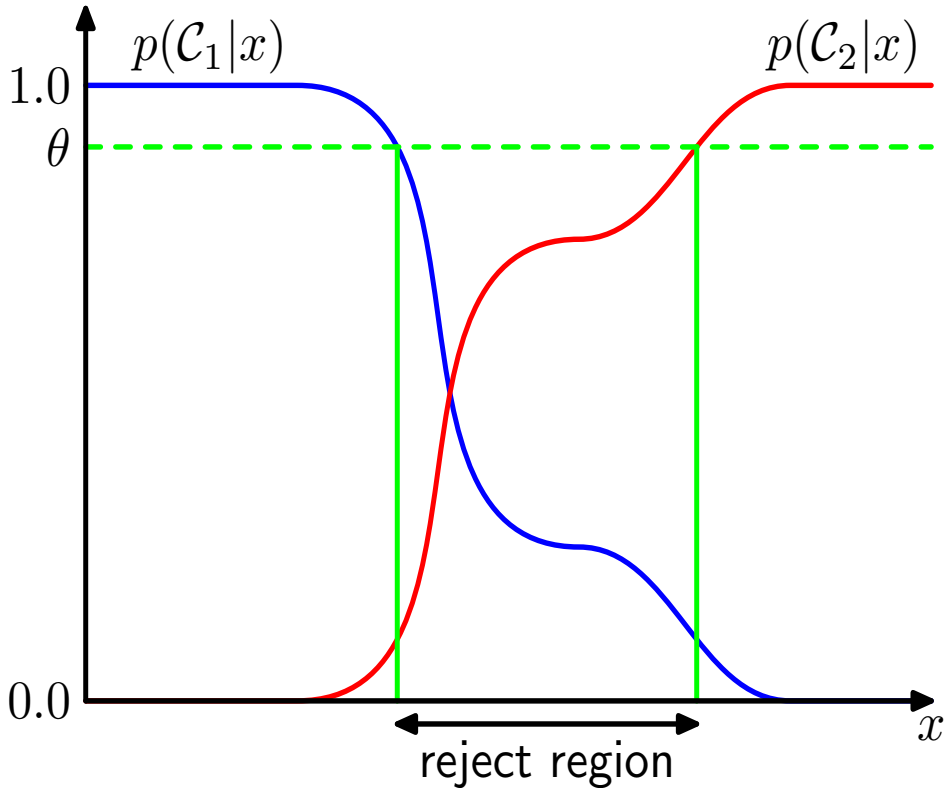
$$\mathbb{E}(L) = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, \mathcal{C}_k) dx$$

Minimizing the expected loss

$$\mathbb{E}(L) = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, \mathcal{C}_k) dx$$

- The goal is to choose the regions $\mathcal{R}_j$ in order to minimize $\mathbb{E}(L)$, which implies that for each x we should minimize $\sum_k L_{kj} p(\mathbf{x}, \mathcal{C}_k)$
- Thus the trivial decision rule that minimizes $E(L)$ assigns each new $\mathbf{x}$ to the class j for which the quantity $\sum_k L_{kj} p(\mathbf{x}, \mathcal{C}_k)$ is minimum.

The reject option



Inputs x such that $\max_k p(\mathcal{C}_k|\mathbf{x}) < \theta$, are rejected.

Inference and decision

There are three distinct approaches to solving decision problems, in decreasing order of complexity,

1. **Generative models**: Learn $p(\mathbf{x}|\mathcal{C}_k)$ for each class $\mathcal{C}_k$) individually. Also separately infer the prior class probabilities $p(\mathcal{C}_k)$. Then use Bayes' theorem to find $p(\mathcal{C}_k|\mathbf{x})$. One may also model the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$ directly.
2. **Discriminative models**: Learn $p(\mathcal{C}_k|\mathbf{x})$, and then subsequently use decision theory to assign each new $\mathbf{x}$ to one of the classes.
3. **Non probabilitic models**: Find a function $f(\mathbf{x})$, called a discriminant function, which maps each input $\mathbf{x}$ directly onto a class label. Probabilities play no role here.

Generative models

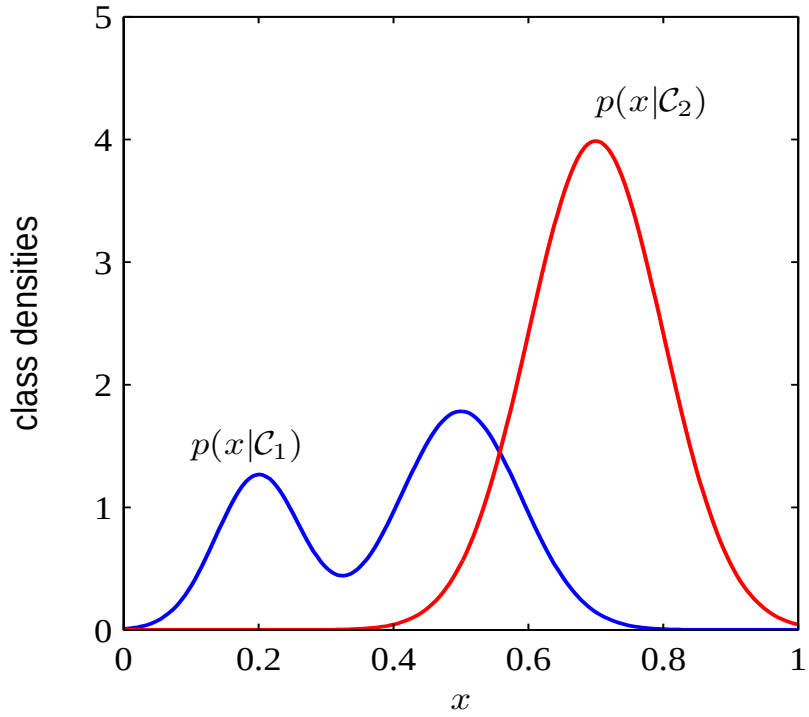
The relative merits of **Generative models**:

- If $\mathbf{x}$ has high dimensionality, a large training set is needed to determine $p(\mathcal{C}_k|\mathbf{x})$ from $p(\mathbf{x}|\mathcal{C}_k)$. to reasonable accuracy.
- Advantage: it allows the marginal density of data to be determined,

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

- Useful for detecting new data points that have low probability under the model known as (*outlier detection*) or (*novelty detection*).
- Disadvantage: wasteful of computational resources and excessively demanding of data if we only wish to make classification decisions. $p(\mathbf{x}|\mathcal{C}_k)$ may contain a lot of structure that has little effect on the posterior. probabilities

Illustration



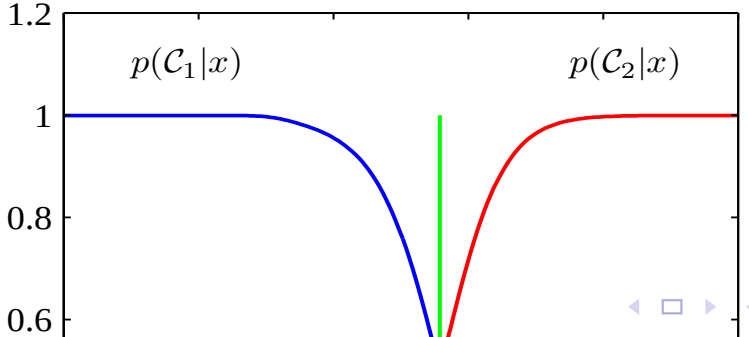
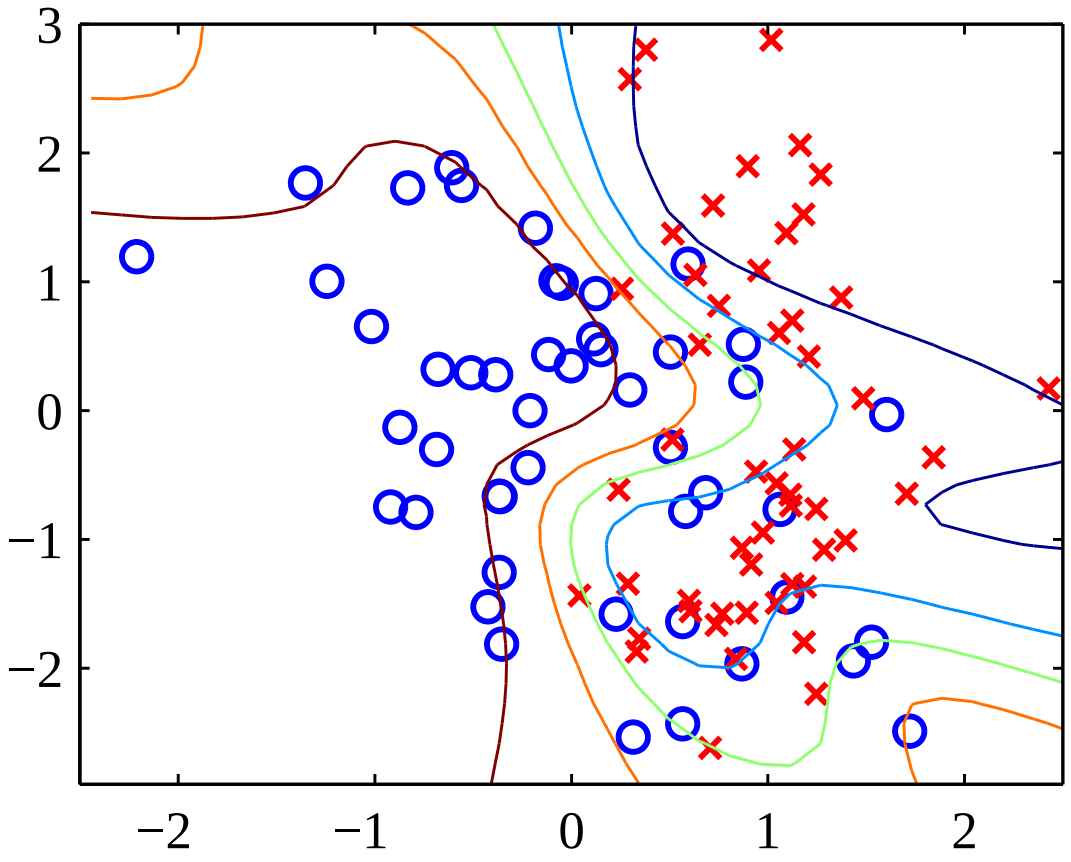


Illustration: Decision boundaries



Outline

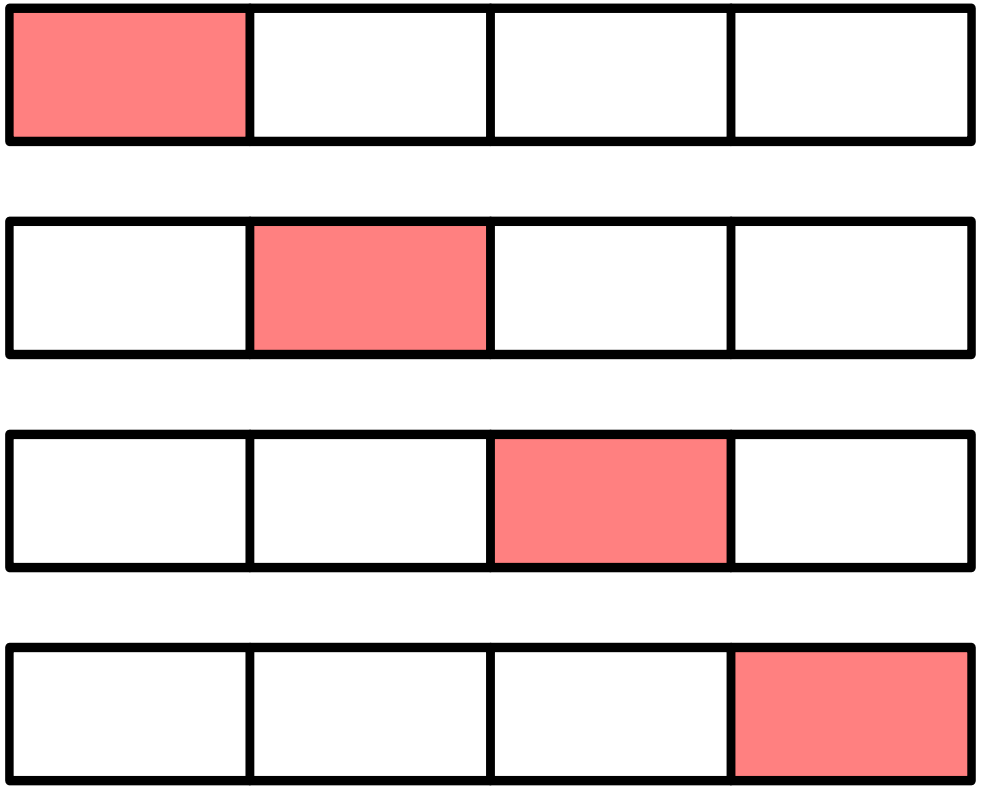
Model Selection

- We have already seen that, in the maximum likelihood approach, the performance on the training set is not a good indicator of predictive performance on unseen data due to the problem of over-fitting
- If data is plentiful, the best approach is simply to decompose $\mathcal{D}$ into 3 subsets:
  - A **training set**, some of the available data to train a range of models.
  - A **validation set** to estimate of the predictive performance of the models and compare them on independent data.
  - A **test set** on which the performance of the selected model is finally evaluated.

Cross-validation

- In many applications, however, the supply of data for training and testing will be limited, and in order to build good models, we wish to use as much of the available data as possible for training.
- One solution is to use S-fold **cross-validation**. The technique involves taking the available data and partitioning it into S groups. Then $S - 1$ of the groups are used to train a set of models that are then evaluated on the remaining group.
- The procedure is repeated for all S possible choices for the held-out group and the performance scores from the S runs are then averaged

Cross-validation



Evaluation of binary classifiers

- There are many metrics that can be used to measure the performance of a classifier or predictor; different fields have different preferences for specific metrics due to different goals.
- To evaluate a classifier, one compares its output to a perfect classification and cross tabulates the data into a 2×2 contingency table (confusion matrix),

	Positive	Negative
Positive prediction	TP	FP
Negative prediction	FN	TN

- **Prevalence** (i.e., the positive rate) has a significant impact on prediction values. A stupid classifier achieves an accuracy rate above 50% with (*imbalanced data sets*).
- The cost function may not be **symmetrical**, depending on the application.

Evaluation of binary classifiers

$$\text{Precision} = \frac{TP}{TP + FP}; \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}; \quad \text{Sensibility} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}; \quad \text{Balanced Acc.} = \frac{1}{2}(\text{Spe.} + \text{Sens.})$$

Each of the 2×2 tables can be summarized as a pair of 2 numbers. In addition to the paired metrics, there are also single metrics that give a single score. An F-score is a combination of the precision and the recall:

$$\text{F-score} = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

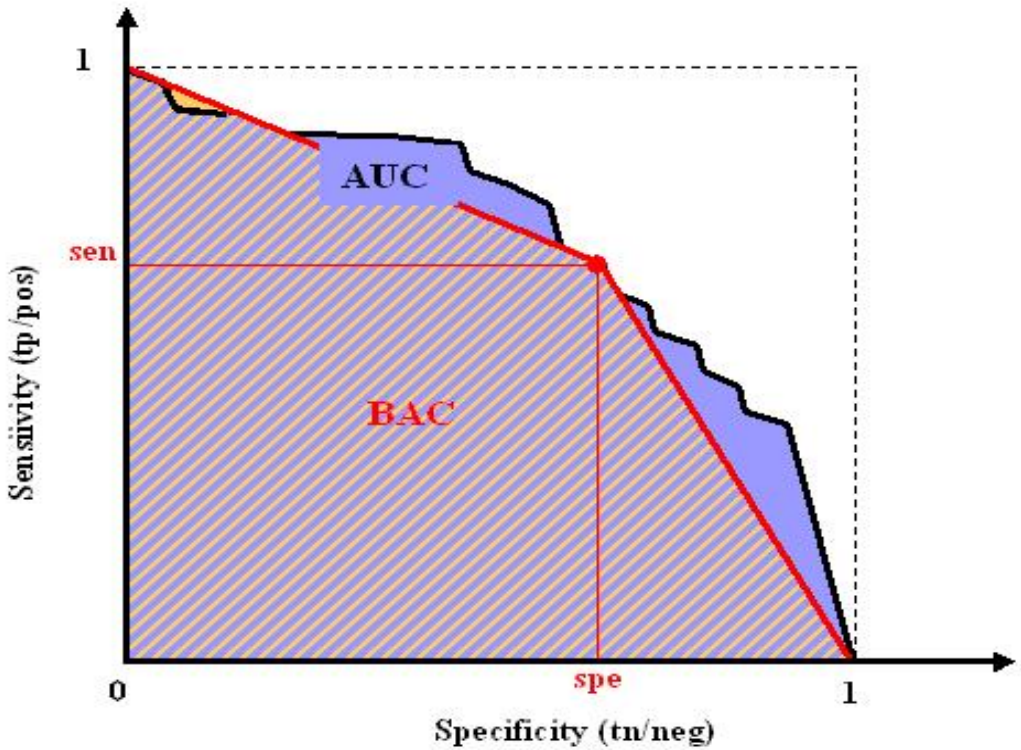
Evaluation of binary classifiers

- **Prevalence** has a significant impact on prediction values.
- Suppose there is a test for a disease with 99% sensitivity and 99% specificity. If 2000 people are tested and the prevalence is 50%, 1000 of them are sick and 1000 healthy. Thus about 990 TP (and 990 TN) are likely, with 10 FP (and 10 FN). The P and N prediction values would be 99%.
- If the prevalence is only 5%, only 100 are sick. The likely result is 99 TP, 1 FN, 1881 TN and 19 FP. Of the 19+99 people tested positive, only 99 really have the disease, - only 84% chance that a patient has the disease given that his test result is positive, and only 1 chance in 1882, (0.05%) that the patient has the disease despite a negative test result.

ROC Curve

- The overall performance of the classifier can be visualized and studied using the Receiver Operating Characteristic curve of **ROC curve**.
- A ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination **threshold** is varied. The curve is created by plotting the sensitivity against the specificity at various threshold settings.
- The best possible prediction method would yield a point in the upper left corner or coordinate $(1, 1)$ of the ROC space,
- The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random), points below the line represent poor results (worse than random)

Courbe ROC



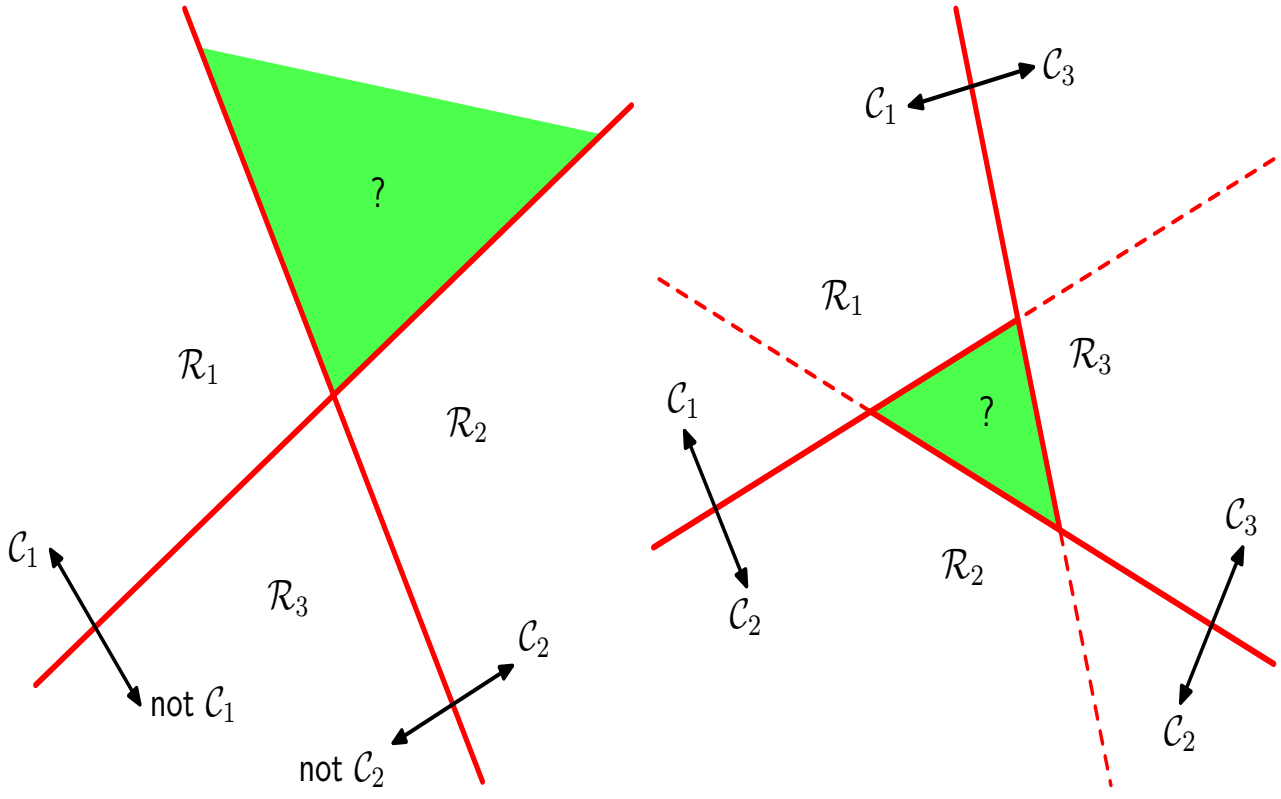
The area under the curve (AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

Outline

Multiclass classification

- Multiclass or multinomial classification is the problem of classifying instances into one of K classes ($K > 2$).
- Many binary classification algorithms can be turned into multiclass classifiers by a variety of strategies.
  1. *One-against-all*: involves training a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. Base classifiers produce real-valued confidence scores rather than just a class label.
  2. *One-against-one*: $K(K-1)/2$ binary classifiers are trained, each with the samples of the pair of classes. At prediction time, all classifiers are applied; the class with the highest number of votes is usually predicted.
  3. *Error-Correcting Output Codes* (ECOC) is an ensemble method designed for multi-class classification problem where each class is assigned a unique binary string of length n, called a codeword.

Illustration



ECOC - Illustration

Classe	vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	1	1	0	0
9	0	0	0	1	1	0

Table: A 6 bit error-correcting output code for a ten-class problem

ECOC - Illustration

- In digit recognition task, we need to map each hand written digit to one of $K = 10$ classes. Every column is given an interpretation

Column	notation	meaning
1	vl	contains a vertical line
2	hl	contains a horizontal linee
3	dl	contains a diagonal line
4	cc	contains a closed curve
5	ol	contains an open curve on the right side
6	or	contains an open curve on the left side

ECOC - Illustration

- During training, one binary classifier is learned for each column.
- To classify a new data point x, all n binary classifiers are evaluated to obtain a 6-bit string. Finally, we choose the class whose *codeword* is closet to x's output string as the predicted label.
- Example : 110001 is closest to 110000, thus the output class is 4.
- The rows have more bits than is necessary ($\log_2(10)$). Using some redundant "error-correcting" bits, we can tolerant some error.
- If d is the minimum Hamming distance between any pair of code words, then the code can correct at least $[(d-1)/2]$ single bit errors.
- In the previous code, $d = 2$ thus no error is allowed.

ECOC

- There are many ways to design the error-correcting output code.
- When K is small, one can use exhaustive codes. Each code has length $2^{K-1} - 1$. Row 1 contains only ones. Row 2 consists of 2^{K-2} zeros followed by $2^{K-2} - 1$ ones, and so on. This code has the largest inter-row Hamming distance.
- When K is large, random codes can be used. Random code works as well as optimally constructed code.
- The major benefit of error-corrective coding is variance reduction via model averaging.

One-against-one

- One-against-one suffers from ambiguities (just as one-against-all) in that some regions of its input space may receive the same number of votes.
- Instead of taking the class with the highest number of votes, it is possible to combine the $K(K-1)/2$ outputs values of the *One-against-one* strategy to obtain a posterior class probabilities.
- Let $\mathcal{C}_{ij}$ denotes "x is in class $\mathcal{C}_i$ or $\mathcal{C}_j$" and let

$$P_{ij} = P(\mathcal{C}_i|\mathcal{C}_{ij}, X = x)$$

- Can $P(\mathcal{C}_i|X = x)$ be written as a function of P_{ij} ?

One-against-one

- The idea:

$$
\begin{aligned}
P(\bigcup_{j=1}^{K} \mathcal{C}_j | X = x) &= 1 \\
&= P(\bigcup_{j=1, i \neq j}^{K} \mathcal{C}_{ij} | X = x) \\
&= \sum_{j=1, i \neq j}^{K} P(\mathcal{C}_{ij} | X = x) - (K-2) \cdot P(\mathcal{C}_i | X = x)
\end{aligned}
$$

with

$$
P_{ij} = P(\mathcal{C}_i | \mathcal{C}_{ij}, X = x) = \frac{P(\mathcal{C}_i | X = x)}{P(\mathcal{C}_{ij} | X = x)}
$$

One-against-one

- One obtains the K a posteriori probabilities given the $K(K-1)/2$ probabilities P_{ij} :

Recombination

$$P(\mathcal{C}_i|X=x) = \frac{1}{\sum_{j=1,i\neq j}^{K} \frac{1}{P_{ij}} - (K-2)}$$

- With $P_{ij} = 1, \forall i$, we get $P(\mathcal{C}_i|X=x) = 1$ as expected.
- No clear advantage between *one-against-one* and *one-against-all* techniques.

Outline

Multi-label classification

- The multi-label classification (MLC) problem differs from conventional classification, as multiple labels can be assigned simultaneously to the same instance.
- This situation is encountered in many recent real-world problems, including image indexing and annotation, facial expression analysis, text categorization, sentiment analysis, fault-control, drug side effects prediction, genome-wide protein function assignment, early detection of chronic diseases to cite just a few.
- For example, a document can be classified as both "spam" and "politics". Or, a patient can be diagnosed with both "cancer" and "diabetes".
- Multilabel learning is a challenging problem because it is difficult to learn the relationships between multiple labels.
- The MLC problem has received increasing attention from the ML community.

Learning setting

- Learning in MLC amounts to finding a mapping from a space of features $\mathcal{X} = \mathbb{R}^d$ to a space of labels $\mathcal{Y} = \{0, 1\}^m$, given a set of training samples in $\mathcal{X} \times \mathcal{Y}$ and a loss function L.
- Formally, the training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{s}$ consists in i.i.d. samples drawn from the joint distribution $p(\mathbf{x}, \mathbf{y})$ ($\mathbf{x} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$), and the loss function $L : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ is a distance on the label space.
- The goal of learning is then to build a function $\mathbf{h} : \mathcal{X} \to \mathcal{Y}$ which maps any new input $\mathbf{x}$ to its proper set of labels, as closely as possible given the loss function L.
- There are many different cost functions that can be used for multilabel learning.

Learning setting

- From a probabilistic perspective, solving this problem amounts to modeling the conditional joint distribution $p(\mathbf{y}|\mathbf{x})$, and inferring Bayes-optimal predictions by minimizing the expected loss, $\mathbf{h}^{\star}(\mathbf{x}) = \underset{\hat{\mathbf{y}}}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{y}|\mathbf{x}} \left[L(\hat{\mathbf{y}}, \mathbf{y}) \right]$.
- Multi-label classification raises a number of computational and statistical challenges, mainly due the size of the output space which grows exponentially with the number of labels. Typically, modeling naively $p(\mathbf{y}|\mathbf{x})$ requires estimating $O(2^m)$ parameters, while inferring a Bayes-optimal prediction for an arbitrary loss function requires $O(2^{2m})$ evaluations of L.
- Label dependencies have to be incorporated into the learning process in order to improve the classification performance.

MLC cost functions

There are many different cost functions that can be used for multilabel learning. Some of the most common cost functions include:

- **Hamming loss**: This is the most common cost function for multilabel learning. It measures the number of labels that are misclassified.
- **Zero-one loss**: This is a stricter cost function than Hamming loss. It measures the number of labels that are misclassified, as well as the number of labels that are not classified at all.
- **F-measure**: This cost function balances the precision and recall of the predictions.

The choice of cost function depends on the specific application. For example, if the application requires a high precision, then the zero-one loss function may be a good choice. If the application requires a high recall, then the F-measure may be a good choice.

Further MLC cost functions

- The **Jaccard index**, is defined as the number of correctly predicted labels divided by the union of predicted and true labels, $\frac{|T \cap P|}{|T \cup P|}$ where P and T are sets of predicted labels and true labels respectively.
- **Precision** is $\frac{|T \cap P|}{|P|}$,
- **Recall** is $\frac{|T \cap P|}{|T|}$,
- F_1 is their harmonic mean

Hamming loss

- Optimizing decomposable loss functions such as the popular Hamming loss requires only to model the marginal distribution of each label.

$$L_H(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y_i = h_i(\mathbf{x})],$$

  where $\mathbb{I}[\cdot]$ is the standard $\{False, True\} \rightarrow \{0, 1\}$ mapping.

- **Binary relevance** is a simple method for multilabel learning. It treats each label as a separate binary classification problem. This means that the model is trained to predict each label independently of the other labels.
- Binary relevance can be inaccurate if the labels are not independent of each other.

The Zero-one loss

- Contrary to decomposable loss functions, optimizing complex performance metrics such as the subset 0/1 loss, the F-measure or the Jaccard index requires to model the joint distribution of the labels (at least to some extent),
- The *subset* 0/1 loss, which generalizes the well-known 0/1 loss from the conventional to the multi-label setting, i.e.,

$$L_S(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \mathbb{I}[\mathbf{y} = \mathbf{h}(\mathbf{x})].$$

- The risk-minimizing prediction for subset 0/1 loss is given by the mode of the conditional distribution $p(\mathbf{y}|\mathbf{x})$, a.k.a. the maximum a posterior probability estimate (MAP), or the most probable explanation (MPE),

$$\mathbf{h}^\star(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}).$$

- Therefore, one iteration through the 2^m label combinations suffices to compute $\mathbf{h}^\star(\mathbf{x})$ for the subset 0/1 loss.

Learning Methods

- Probabilistic classifier chains (PCC) explicitly learn a maximum-likelihood estimate of $p(\mathbf{y}|\mathbf{x})$ with logistic classifiers for MAP inference (computationally expensive).
- Label ranking is a method that ranks the labels for each instance. The top-ranked labels are then considered to be the predicted labels (difficult to train).
- Ensemble learning is a method for combining the predictions from multiple models. This can be done by averaging the predictions, or by using a voting system (computationally expensive).
- Weighted majority voting combines the predictions from multiple binary classifiers. The weights for the classifiers are determined by their accuracy (computationally expensive).

Challenges of multilabel learning

- There are many challenges associated with multilabel learning. Some of the most common challenges include:
- The curse of dimensionality: This is the problem of having too many features relative to the number of instances. This can make it difficult to learn the relationships between the features and the labels.
- Label imbalance: This is the problem of having some labels that are much more common than others. This can make it difficult to learn the minority labels.
- Class overlap: This is the problem of having labels that are similar to each other. This can make it difficult to distinguish between the labels.
- These challenges can be addressed using a variety of techniques, such as feature selection,

Outline

Shrinkage methods

- We have seen that regularization control the over-fitting phenomenon by adding a penalty term to the error function,
- This technique, also called ridge regression, is an example of shrinkage method applied to least squares regression, to improve a least-squares estimator. Least-squares solutions having a large number of coefficients different from zero are penalized.
- Only the variables whose impact on the empirical risk is considerable have a coefficient bigger than zero and consequently appear in the fitted linear model.
- Doing shrinkage is therefore an implicit embedded manner of doing feature selection since only a subset of variables contributes to the final predictor.

Ridge regression

- An equivalent way to write the ridge problem is

$$\hat{w}_r = \arg\min_w \sum_{i=1}^{N} (y_i - x_i^T w)^2,$$

$$\text{subject to } \sum_{j=1}^{p} w_j^2 \leq L$$

- The ridge regression solution is

$$\hat{w}_r = (X^T X + \lambda I)^{-1} X^T Y$$

where I is the identity matrix of size p

Lasso

- Another well known shrinkage method is lasso where the estimate of the linear parameters is returned by

$$\hat{w}_r = \arg\min_w \sum_{i=1}^{N} (y_i - x_i^T w)^2,$$

$$\text{subject to } \sum_{j=1}^{p} |w_j| \leq L$$

- The 1-norm penalty allows a stronger constraint on the coefficients, however it makes the solution nonlinear and requires a quadratic programming algorithm.
- If $L > \sum_{j=1}^{p} \hat{w}_j$ the lasso returns the common least-squares solution. The penalty factor L is typically set by cross-validation.

Naive Bayes classifier

- The **Naive Bayes** (NB) classifier is a baseline model just as decision trees.
- Consider a multi-class learning problem. Y that takes values in the set $\{c_1, \ldots, c_K\}$. The Bayes optimal classifier should return

$$c^*(x) = \arg \max_{j=1,\ldots,K} \text{Prob}\{Y = c_j | x\}$$

- Using the Bayes theorem,

$$\begin{aligned} c^*(\mathbf{x}) &= \arg \max_{j=1,\ldots,K} \frac{\text{Prob}\{\mathbf{x} | Y = c_j\}\text{Prob}\{Y = c_j\}}{\text{Prob}\{\mathbf{x}\}} \\ &= \arg \max_{j=1,\ldots,K} \text{Prob}\{\mathbf{x} | Y = c_j\}\text{Prob}\{Y = c_j\} \end{aligned}$$

Naive Bayes classifier

- It is easy to estimate each of the a priori probabilities $\text{Prob}\{Y = c_j\}$. The estimation of $\text{Prob}\{\mathbf{x}\{Y = c_j\}$ is much harder.
- The NB classifier is based on the **simplifying assumption** that the input values are **conditionally independent** given the target value:

$$\text{Prob}\{\mathbf{x}|Y = c_j\} = \text{Prob}\{x_1, \ldots, x_n|Y = c_j\} = \prod_{h=1}^{n} \text{Prob}\{x_h|Y = c_j\}$$

- The NB classification is then:

$$c_{NB}(\mathbf{x}) = \arg \max_{j=1,\ldots,K} \text{Prob}\{Y = c_j\} \prod_{h=1}^{n} \text{Prob}\{x_h|Y = c_j\}$$

Naive Bayes classifier

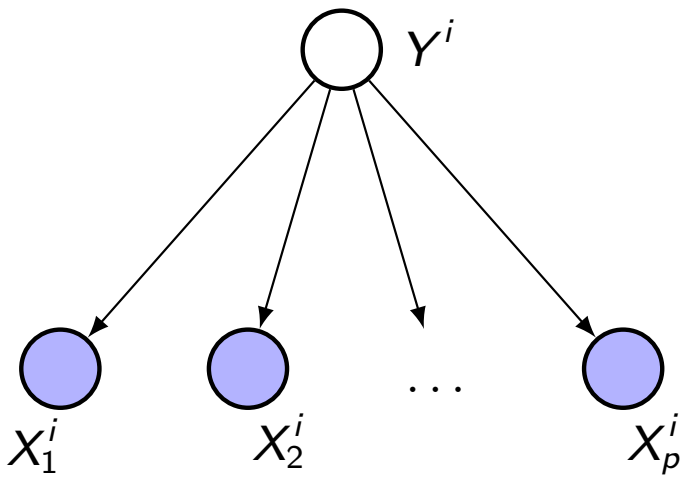
- The naive Bayes classifier is relevant when modeling the joint distribution of $p(x|y)$ is difficult.
- Example : classification of documents based on a bag-of-word representation; i.e. each word of a reference dictionary is present in the document or not.
- The document i is represented by a vector of binary random variables. $X^i : \Omega \mapsto \{0,1\}^p$, with $x_j^i = 1$ iff word j of the dictionary is present in the ith document.

Naive Bayes classifier

- It is possible to approach the problem using directly a *conditional model* of $p(y \mid x)$ or using a *generative model* of the joint distribution modeling separately $p(y)$ and $p(x|y)$ and computing $p(y|x)$ using Bayes rule.
- The naive Bayes model is a generative model.
- Y^i is naturally modeled as a multinomial distribution with $p(y^i) = \prod_{k=1}^{K} \pi_k^{y_k^i}$. However $p(x^i|y^i) = p(x_1^i, \ldots, x_p^i|y^i)$ has a priori $2^p - 1$ parameters. The key assumption made is that $X_1^i, \ldots, X_p^i$ are all independent conditionally on Y^i.
- While ignoring the correlations between words it works well in practice.

Naive Bayes classifier

- These conditional independence assumptions correspond to the following graphical model:



Naive Bayes classifier

- The distribution of Y^i is a multinomial distribution which we parameterize with $(\pi_1, ..., \pi_K)$, and we write

$$\mu_{jk} = P(X_j^{(i)} = 1 | Y_k^{(i)} = 1)$$

- We then have

$$\begin{aligned} p(X^i = x^i, Y^i = y^i) &= p(x_i, y_i) \\ &= p(x^i | y^i) p(y^i) \\ &= \prod_{j=1}^{p} p(x_j^i | y^i) p(y^i) \end{aligned}$$

Naive Bayes classifier

- The last expression leads to

$$p(x^i, y^i) = \left[\prod_{j=1}^{p}\prod_{k=1}^{K} {\mu_{jk}}^{x_j^i y_k^i} (1-\mu_{jk})^{(1-x_j^i)y_k^i}\right] \prod_{k=1}^{K} \pi_k^{y_k^i}$$

$$\log p(x^i, y^i) = \sum_{k=1}^{K}\left(\sum_{j=1}^{p}\left(x_j^i y_k^i \log \mu_{jk} + (1-x_j^i) y_k^i \log(1-\mu_{jk})\right) + y_k^i \log(\pi_k)\right)$$

- Note that, in spite of the name the naive Bayes classifier is not a Bayesian approach to classification.

Support vector machines (linear case)

- Consider a binary classification task. The problem of separating input data by a linear boundary is ill-posed as there are infinitely many possible separating hyperplanes characterized by the equation

$$\beta_0 + x^T\beta = 0$$

- The vector normal to the hyperplane is given by $\beta^* = \frac{\beta}{|\beta|}$
- The signed distance of a point x to the hyperplane is called the geometric margin and is given by

$$\beta^{*T}(x - x_0) = \frac{x^T\beta - \beta x_0^T}{|\beta|} = \frac{1}{|\beta|}(x^T\beta + \beta_0)$$

Support vector machines (linear case)

- This technique relies on an optimization approach to compute separating hyperplanes and was shown to lead to good classification performance on real data.
- The **SVM approach computes the (unique) maximal margin hyperplane** for a training set, i.e. the optimal separating hyperplane which separates the two classes by maximizing the distance to the closest point from either class.
- The problem is modeled as the optimization problem

$$\max_{\beta,\beta_0} C \text{ subject to } \quad \frac{1}{|\beta|} y_i(x_i^T \beta + \beta_0) \geq C \quad \text{for } i = 1, \ldots, N$$

So that all the points are at least a distance C from the decision boundary.

Margin

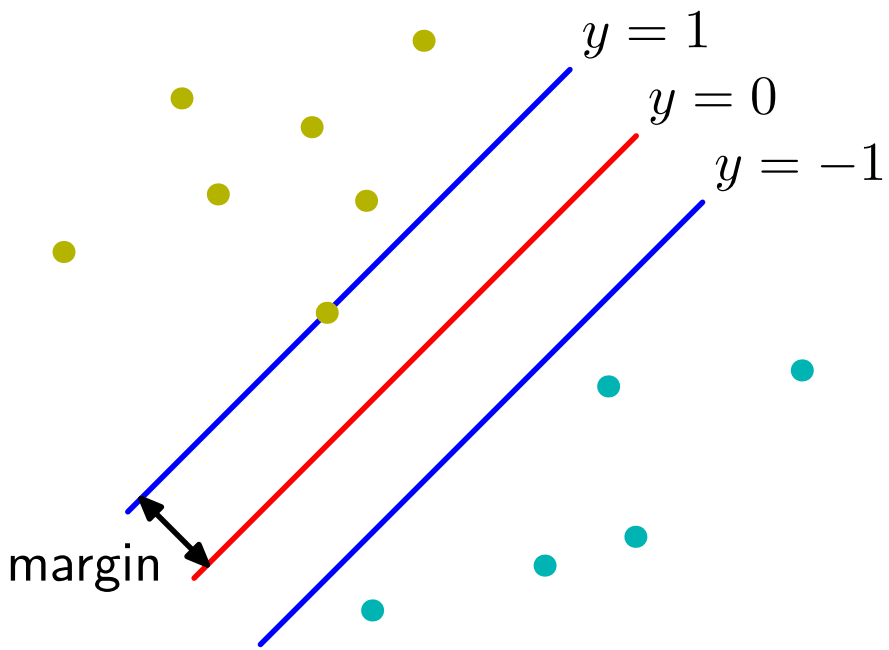


Figure: The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points.

Maximizing the margin

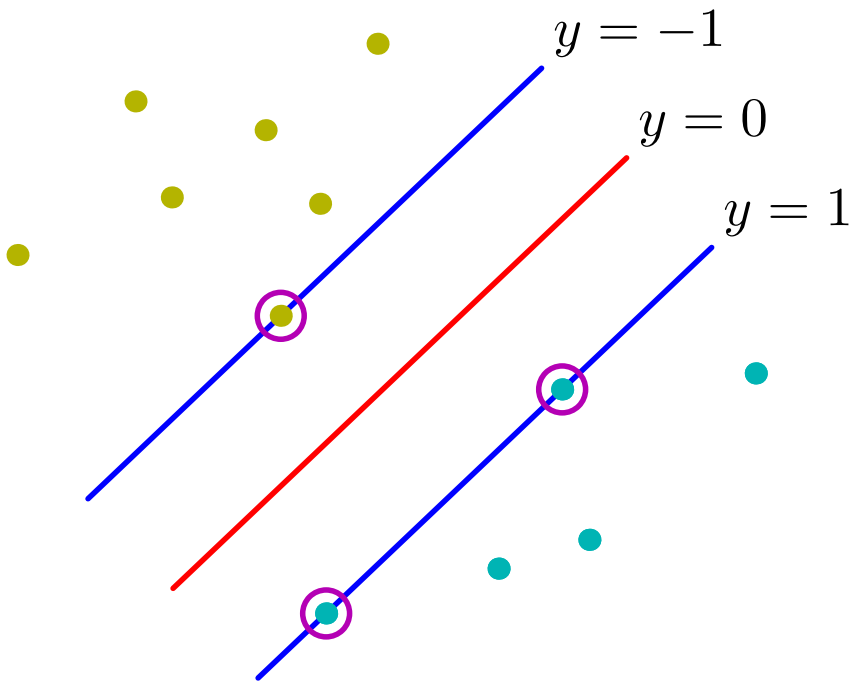


Figure: The location of this boundary is determined by a subset of the data points, known as support vectors.

Support vector machines

- We can set $|\beta| = 1/C$. The maximization problem can be reformulated in a minimization form

$$\min_{\beta,\beta_0} \frac{1}{2}|\beta|^2 \text{ subject to } \qquad y_i(x_i^T\beta + \beta_0) \geq 1 \quad \text{for } i = 1, \ldots, N$$

- The constraints impose a margin around the linear decision of thickness $1/|\beta|$.
- This optimization problem is a **convex optimization problem** (quadratic criterion with linear inequality constraints) where the primal Lagrangian is

$$L_P(\beta, \beta_0) = \frac{1}{2}|\beta|^2 - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - 1]$$

$\alpha_i \geq 0$ are the Lagrangian multipliers.

Support vector machines

- Setting the derivatives β and β_0 to zero we obtain:

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, \qquad 0 = \sum_{i=1}^{N} \alpha_i y_i$$

- Substituting these in the primal form, we obtain the dual

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k$$

subject to $\alpha_i \geq 0$.

Support vector machines

- The dual optimization problem is now

$$\max_{\alpha} L_D = \max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \langle x_i, x_k \rangle$$

  subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad \text{for} \quad \alpha_i \geq 0, \quad i = 1, \ldots, N$$

  .

- $\langle x_i, x_k \rangle$ is the inner product of x_i and x_k.

Support vector machines

- It can be shown that the optimal solution must satisfy the Karush-Kuhn-Tucker (KKT) condition

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - 1] = 0, \qquad \forall i$$

- This means that either of these two situations holds
  1. $y_i(x_i^T\beta + \beta_0) = 1$, i.e. the point is on the boundary of the margin, then $\alpha_i > 0$
  2. $y_i(x_i^T\beta + \beta_0) > 1$, i.e. the point is not on the boundary of the margin, then $\alpha_i = 0$
- The training points having an index i such that $\alpha_i > 0$ are called the **support vectors**.

Support vector machines

- The **decision function** can be written as

$$h(x, \beta, \beta_0) = \text{sign}[x^T \beta + \beta_0] = \text{sign}[\sum_{\text{SV}} y_i \alpha_i \langle x_i, x \rangle + \beta_0]$$

- Attractive property of SVM: it is expressed as a function of a limited number training points (support vectors) which are on the boundaries. Points far from the class boundary do not play a major role, unlike linear discriminant models.
- In the separable case

$$C = \frac{1}{|\beta|} = \frac{1}{\sqrt{\sum_{i=1}^{N} \alpha_i}} \quad (1)$$

Support vector machines (nonlinear case)

- The extension of the SVM to nonlinear classification relies on the transformation of the input variables input space by using **basis functions**.
- Consider a regression problem where $x \in \mathcal{X} \subset \mathbb{R}^n$ and define m new transformed variables $z_j = z_j(x), j = 1, \ldots, m$, where $z_j(\cdot)$ is a pre-defined nonlinear transformation (e.g. $z_j(x_1, x_2) = \log x_1 + \log x_2$). This is equivalent to mapping the input space $\mathcal{X}$ into a new space, known as the **feature space**, $\mathcal{Z} = \{z = z(x) | x \in \mathcal{X}\}$,
- If $m < n$, this boils down to a dimensionality reduction.
- We may fit a linear model $y = \sum_{j=1}^{m} \beta_m z_m$ to the training data in the new input space $z \in \mathbb{R}^m$. By doing this, we carry out a **nonlinear fitting** of data using conventional linear techniques.

Kernels

- A **dot-product kernel** is a function K, such that for all $x, x' \in \mathcal{X}$

$$K(x, x') = \langle z(x), z(x') \rangle$$

where $\langle z_1, z_2 \rangle = z_1^T z_2$ stands for the inner product and $z(\cdot)$ is the mapping from the original to the feature space $\mathcal{Z}$.

- Consider a simple kernel function given by $K(x, x') = \langle x, x' \rangle^2 = \langle z(x), z(x') \rangle$. The feature mapping takes the form $z(x) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^T$ and therefore comprises second order terms.
- Stationary kernels $K(x, x') = k(x - x')$ are invariant to translations in input space, e.g. radial basis functions or Gaussian kernels.
- Extensions of kernels to handle symbolic objects greatly expand the range of problems that can be addressed.

Support vector machines

- The parametric identification step for binary classification by SVM in the separable case requires the solution of a quadratic programming problem in the space $\mathcal{Z}$:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k \langle z_i, z_k \rangle$$

$$= \max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k K(x_i, x_k)$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0, \text{ and } \alpha_i \geq 0, \quad i = 1, \ldots, N$$

Support vector machines

- The resolution of this problem differs from the linear one by the replacement of the quantities $\langle x_i, x_k \rangle$ with $K(x_i, x_k)$
- Whatever the dimensionality m, the SVM computation requires only the availability of the kernel matrix K.
- We don't need to know the underlying feature transformation function $z(x)$.
- The use of a kernel function is an attractive computational short-cut. In practice, the approach consists in defining a kernel function directly, hence implicitly defining the feature space.

Support vector machines

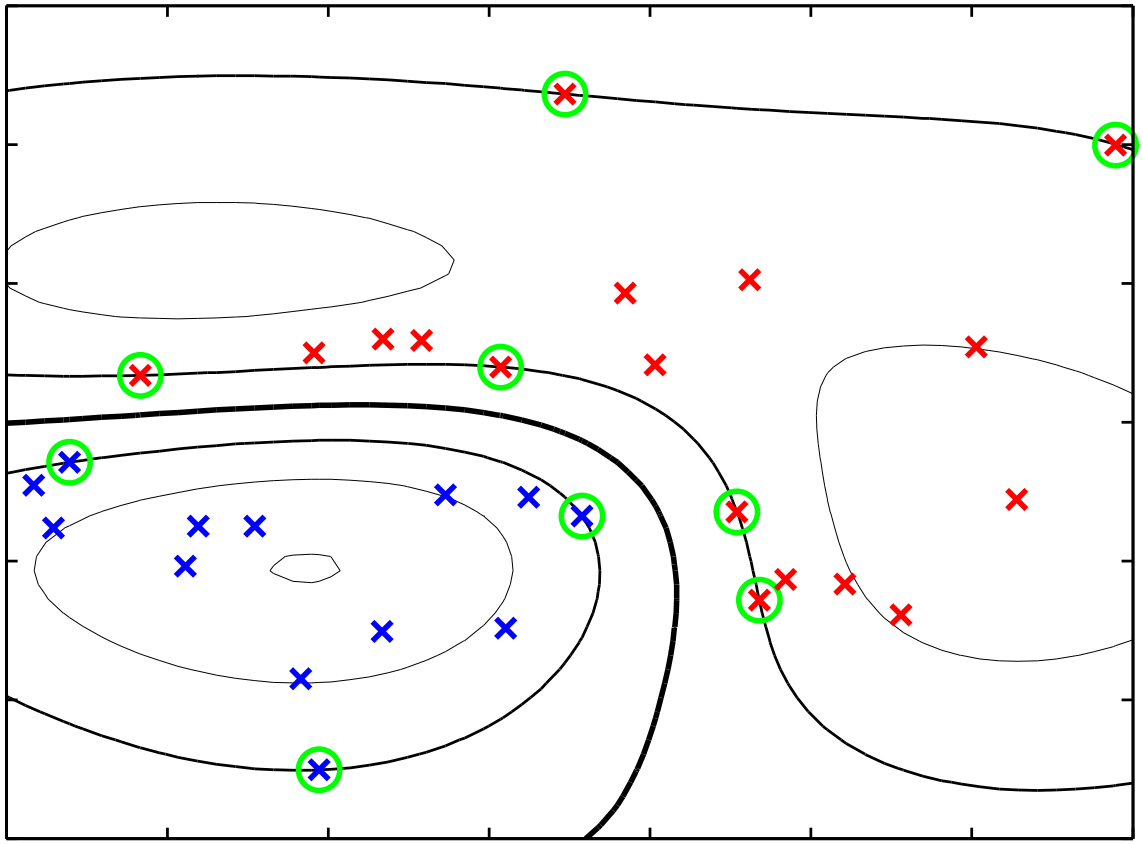


Figure: Binary classification in two dimensions showing contours of constant $f(x)$ obtained from a SVM with Gaussian kernel. The support vectors are in circles.

Support vector machines

- Although the data set is not linearly separable in the two-dimensional data space $\mathcal{X}$, it is linearly separable in the nonlinear feature space $\mathcal{Z}$ defined implicitly by the nonlinear kernel function K.
- Thus the training data points are perfectly separated in the original data space.

Overlapping class distributions

- So far, we assumed that the training data points are **linearly separable** in the feature space $\mathcal{Z}$.
- In practice, the class-conditional distributions may overlap, in which case exact separation of the training data can lead to poor generalization.
- We need to reformulated the maximization problem so as to allow some of the training points to be misclassified. The goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary,

$$\min_{\beta,\beta_0} \frac{1}{2}|\beta|^2 + C\sum|\xi_i| \text{ subject to } y_i({x_i}^T\beta + \beta_0) \geq 1 - \xi_i, \forall i$$

- The hyperparameter $C > 0$ controls the trade-off between the slack variable penalty and the margin.

Overlapping class distributions

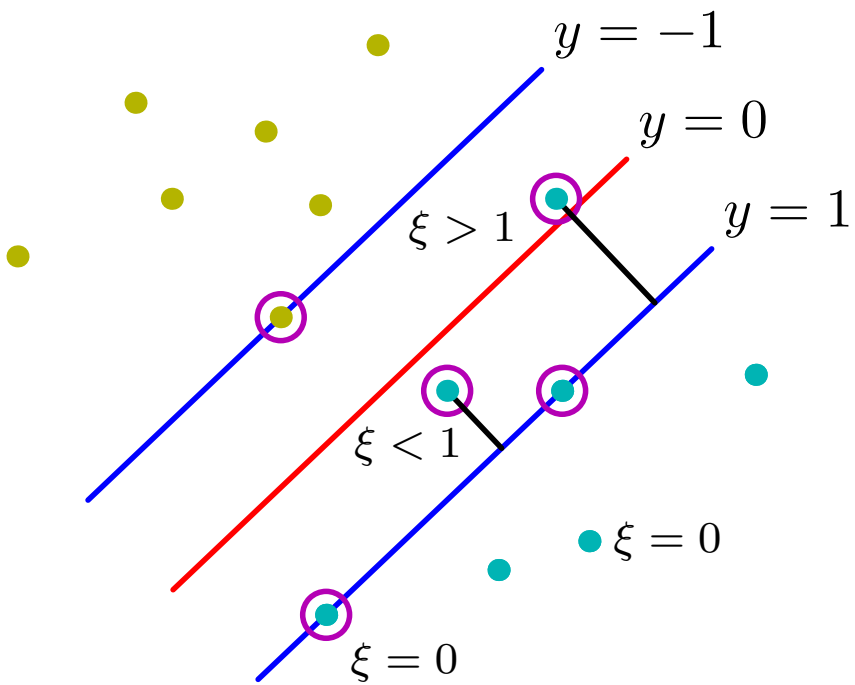


Figure: Illustration of the slack variables. Relaxing the hard margin constraint gives a soft margin and allows some of the training set data points to be misclassified.

Overlapping class distributions

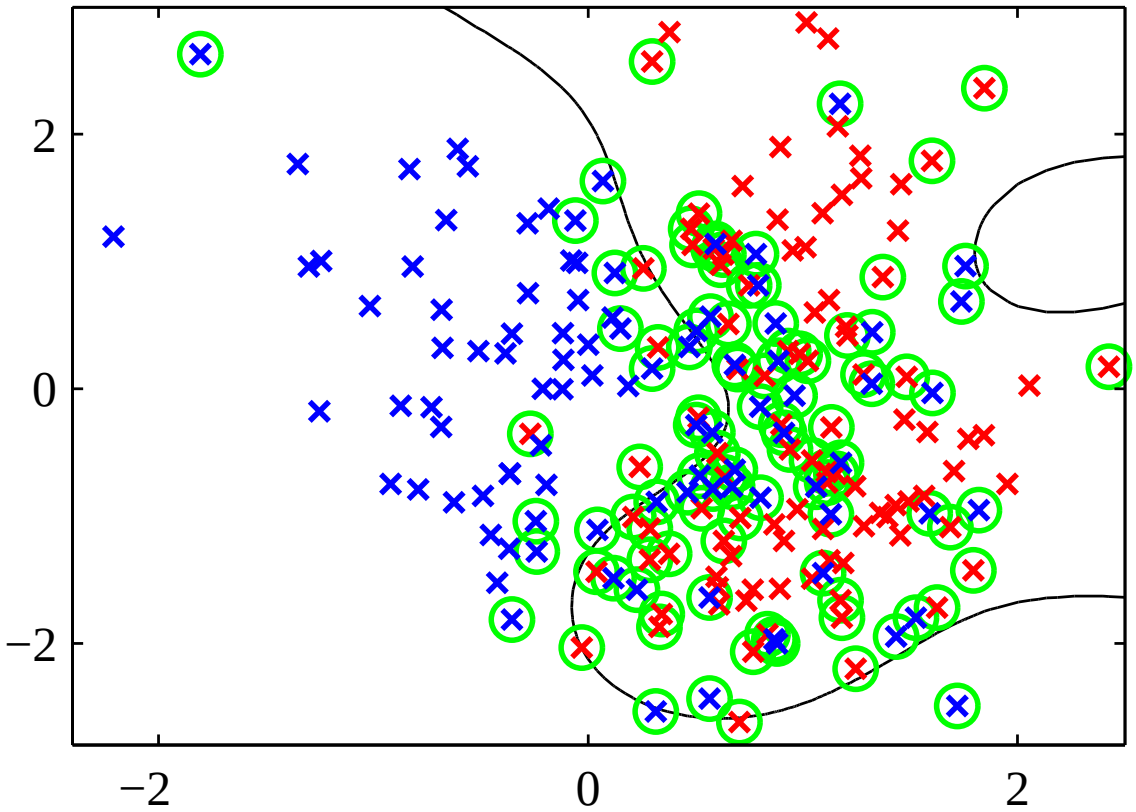


Figure: Illustration of the SVM with Gaussian kernels applied to a non separable data set in two dimensions.

Decision and Regression Trees

- **Tree induction** from samples have been an active topic in the machine learning, the most representative methods of decision-tree induction are the ID3, C4 and the CART (Classification and Regression Trees) algorithm.
- A decision tree partitions the input space into **mutually exclusive regions**, each of which is assigned a procedure to characterize its data points
- An internal node is a decision-making unit that evaluates a decision function to determine which child node to visit next. A terminal node or leaf has no child nodes and is associated with one of the partitions of the input space.
- In classification trees each terminal node contains a label that indicates the class for the associated input region.

Classification and Regression Trees

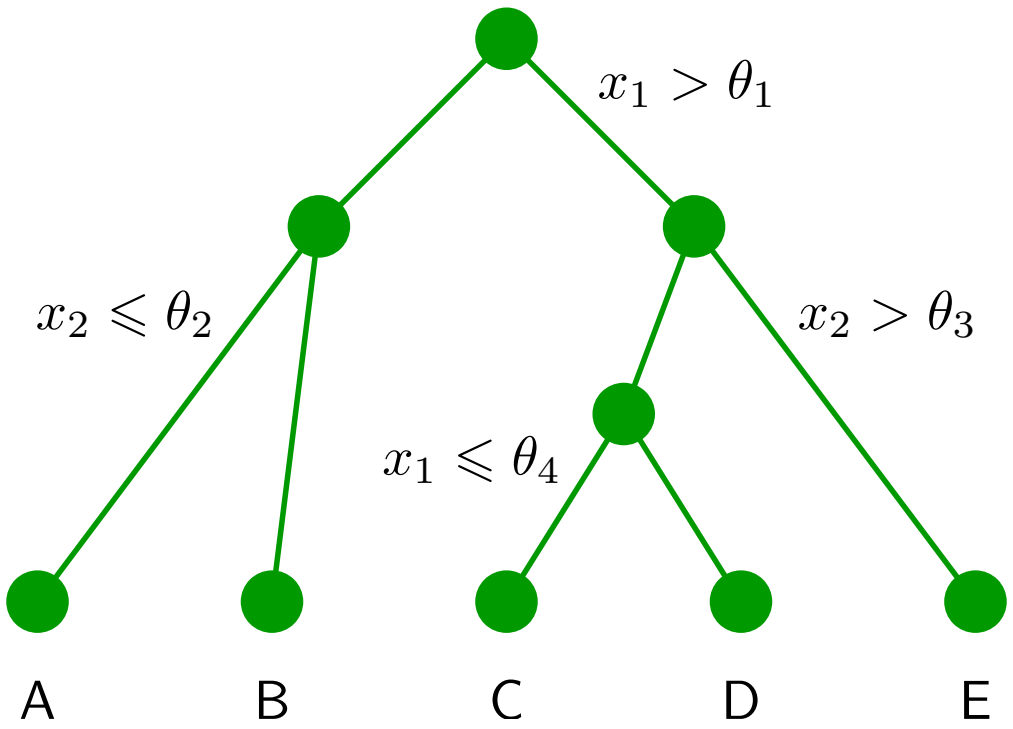


Figure: A binary decision tree.

Decision and Regression Trees

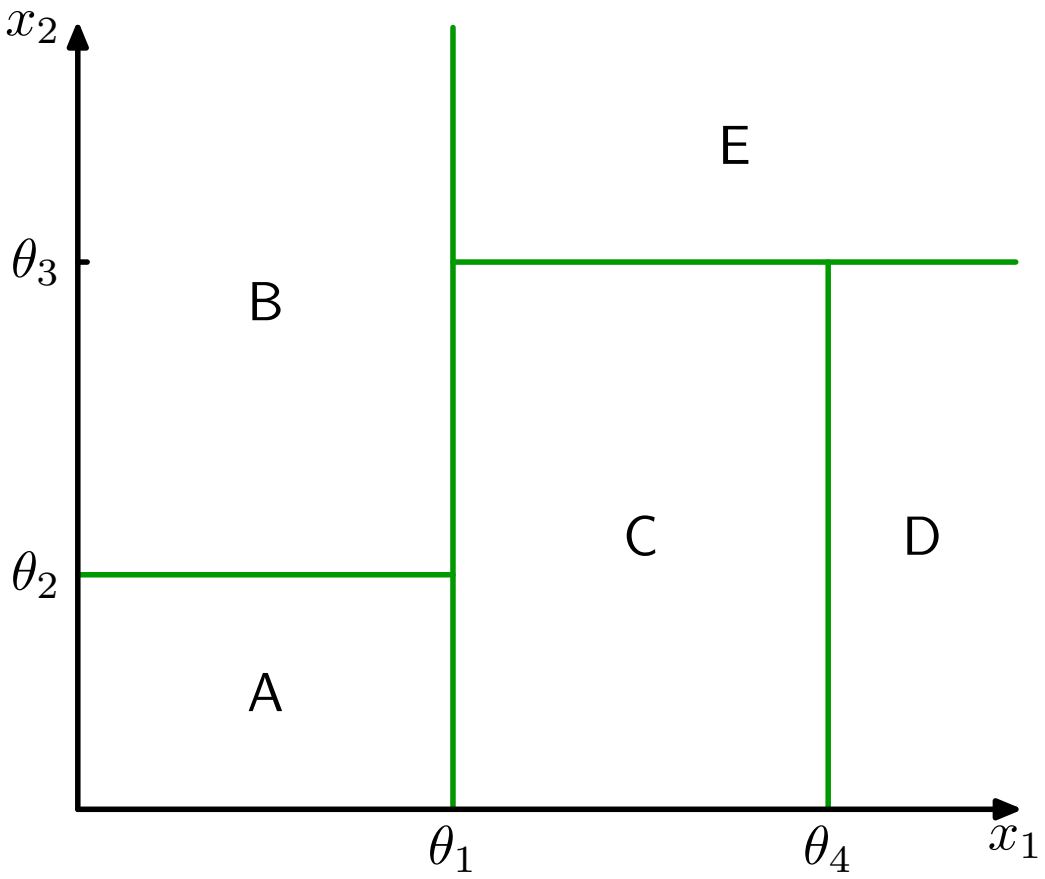


Figure: Input space partitioning induced on the input space by the binary tree.

Decision and Regression Trees

- The structural identification in binary regression trees addresses the problem of choosing the optimal partitioning of the input space.
- Two steps of the procedure to construct an appropriate decision tree: CART first grows the tree on the basis of the training set, and then prunes the tree back based on a minimum cost-complexity principle.
- Tree growing : CART makes a succession of splits that partition the training data into disjoint subsets. Starting from the root node that contains the whole dataset, an exhaustive search is performed to find the split that best reduces a certain cost function.

Regression Trees

- Consider a node t and let $D(t)$ be the corresponding subset of the original D_N. The empirical error of the local model fitting the $N(t)$ data contained in the node t is:

$$R_{emp}(t) = \min_{\alpha_t} \sum_{i=1}^{N(t)} L(y_i, h_t(x_i, \alpha_t))$$

- For any possible split s of node t into the two children t_r and t_l (and $N(t_r) + N(t_l) = N(t)$), define

$$\Delta E(s, t) = R_{emp}(t) - (R_{emp}(t_l) + R_{emp}(t_r))$$

- **The best split** is $s^* = \arg\max_s \Delta E(s, t)$

Decision Trees

- Let $p(\mathcal{C}_k)$ the probability of class $\mathcal{C}_k$ in some node.
- Heterogeneity in the candidate node can measured by several criteria:
  - **Entropy** $(ID3, C4.5)$: $H = -\sum_k p(\mathcal{C}_k) \log_2(p(\mathcal{C}_k))$, bounded by $\log_2(nbclasses)$ with equally balanced classes.
  - **Gini index** (CART) : $Gini = 1 - \sum_k p^2(\mathcal{C}_k)$
  - **Error rate** : $Error = 1 - \max_k(p(\mathcal{C}_k))$
- For example, with the Gini index at node t, $Gini(t)$, the **homogeneity gain** of a split s at node t is given by,

$$\Delta E(s,t) = Gini(t) - \sum_{j=1}^{m} p(t_j) Gini(t_j)$$

where $p(t_j)$ is the probability of reaching node t_j from node t.

Decision and Regression Trees

- Once the best split is attained, the dataset is partitioned into the two disjoint subsets of length $N(t_r)$ and $N(t_l)$, respectively. The same method is **recursively applied** to all the leaves.
- The procedure terminates either when the error measure associated with a node falls below a certain tolerance level, or when the error reduction ΔE resulting from further splitting does not exceed a threshold value.
- The tree that the growing procedure yields is typically too large and presents a serious risk of overfitting the dataset. For that reason a **pruning procedure** is often adopted.

Tree pruning

- Consider a fully expanded tree T_{max} characterized by L terminal nodes. Let us introduce a complexity based measure of the tree performance

$$R_\lambda(T) = R_{emp}(T) + \lambda|T| \tag{2}$$

where λ is a hyper-parameter that accounts for the tree's complexity and $|T|$ is the number of terminal nodes of the tree T. For a fixed λ we define with $T(\lambda)$ the tree structure which minimizes $R_\lambda(T)$

- λ is gradually increased in order to generate a sequence of tree configurations with decreasing complexity

$$T_L = T_{max} \supset T_{L-1} \supset \cdots \supset T_2 \supset T_1 \tag{3}$$

where T_i has i terminal nodes.

Tree pruning

- In practice, this requires a **sequence of shrinking steps** where for each step we select the value of λ leading from a tree to a tree of inferior complexity.
- At the end of the shrinking process we have a sequence of candidate trees which have to be properly assessed in order to perform the structural selection.
- Cross-validation or independent testing can be used for model selection.

Regression Trees

- Regression trees are a very easy-to-interpret representation of a nonlinear input/output mapping.
- However, these methods are characterized by rough **discontinuity at the decision boundaries** which might bring undesired effects to the overall generalization.
- Dividing the data by partitioning the input space shows typically small estimator bias but at the cost of an increased variance. This is particularly problematic in high-dimensional spaces where data become sparse.
- One response to the problem is the adoption of simple local models (e.g. constant or linear to minimize variance at the cost of an increased bias) or to use of soft splits, allowing data to lie simultaneously in multiple regions.

Combination of two estimators

- Consider two unbiased estimators $\hat{\boldsymbol{\theta}}_1$ and $\hat{\boldsymbol{\theta}}_2$ of the same parameter θ,

$$E[\hat{\boldsymbol{\theta}}_1] = \theta \qquad E[\hat{\boldsymbol{\theta}}_2] = \theta$$

having equal and non zero variance

$$\text{Var}\left[\hat{\boldsymbol{\theta}}_1\right] = \text{Var}\left[\hat{\boldsymbol{\theta}}_2\right] = v$$

and being uncorrelated, i.e. $\text{Cov}[\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2] = 0$.

- Let $\hat{\boldsymbol{\theta}}_{\text{cm}}$ be the combined estimator

$$\hat{\boldsymbol{\theta}}_{\text{cm}} = \frac{\hat{\boldsymbol{\theta}}_1 + \hat{\boldsymbol{\theta}}_2}{2}$$

Combination of two estimators

- This estimator has the nice properties of being unbiased

$$E[\hat{\theta}_{cm}] = \frac{E[\hat{\theta}_1] + E[\hat{\theta}_2]}{2} = \theta$$

and with a smaller variance than the original estimators

$$\text{Var}\left[\hat{\theta}_{cm}\right] = \frac{1}{4}\text{Var}\left[\hat{\theta}_1 + \hat{\theta}_2\right] = \frac{\text{Var}\left[\hat{\theta}_1\right] + \text{Var}\left[\hat{\theta}_2\right]}{4} = \frac{v}{2}$$

- This trivial computation shows that the simple average of two unbiased estimators with a non zero variance returns a combined estimator with reduced variance.

Combination of m estimators

- Now, we want to estimate the unknown parameter θ by combining a set of m estimators $\hat{\boldsymbol{\theta}}_j, j = 1, \ldots, m.$ having expected values and variances given by,

$$E[\hat{\boldsymbol{\theta}}_j] = \mu_j \quad \text{Var}\left[\hat{\boldsymbol{\theta}}_j\right] = v_j$$

- Suppose we are interested in estimating θ by forming a linear combination

$$\hat{\boldsymbol{\theta}}_{\text{cm}} = \sum_{j=1}^{m} w_j \hat{\boldsymbol{\theta}}_j = w^T \hat{\boldsymbol{\theta}}$$

- $\hat{\boldsymbol{\theta}} = [\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_m]^T$ is the vector of estimators and $w = [w_1, \ldots, w_m]^T$ is the weighting vector.

Combination of m estimators

- The mean-squared error of the combined system is

$$\begin{aligned}\text{MSE} &= E[(\hat{\boldsymbol{\theta}}_{\text{cm}} - \theta)^2] \\ &= E[(w^T\hat{\boldsymbol{\theta}} - E[w^T\hat{\boldsymbol{\theta}}])^2] + (E[w^T\hat{\boldsymbol{\theta}}] - \theta)^2 \\ &= E[(w^T(\hat{\boldsymbol{\theta}} - E[\hat{\boldsymbol{\theta}}]))^2] + (w^T\mu - \theta)^2 \\ &= w^T\Omega w + (w^T\mu - \theta)^2\end{aligned}$$

  where Ω is the covariance matrix whose ij^{th} term is $\Omega_{ij} = E[(\hat{\boldsymbol{\theta}}_i - \mu_i)(\hat{\boldsymbol{\theta}}_j - \mu_j)]$ and $\mu = (\mu_1, \ldots, \mu_m)^T$ is the vector of expected values.

- The MSE error has a variance and a bias terms dependent the covariance and a bias of the single estimators.

Linear constrained combination

- A commonly used constraint is

$$\sum_{j=1}^{m} w_j = 1, \quad w_j \geq 0, \quad j = 1, \ldots, m$$

- This means that the combined estimator is unbiased if the individual estimators are unbiased.
- The constraint can be enforced in minimizing the MSE by writting w as

$$w = (u^T g)^{-1} g$$

where $u = (1, \ldots, 1)^T$ is an m-dimensional vector of ones, $g = (g_1, \ldots, g_m)^T$ and $g_j > 0, \forall j = 1, \ldots, m$.

Linear constrained combination

- The Lagrangian function writes

$$L = w^T \Omega w + (w^T \mu - \theta)^2 + \lambda(w^T u - 1)$$

with λ Lagrange multiplier.

- The optimum is achieved if we set

$$g^* = [\Omega + (\mu - \theta u)(\mu - \theta u)^T]^{-1} u$$

Linear constrained combination

- With unbiased estimators ($\mu = \theta$) we obtain

$$g^* = \Omega^{-1} u$$

- With unbiased and uncorrelated estimators

$$g_j^* = \frac{1}{v_j} \qquad j = 1, \ldots, m$$

This means that the optimal term g_j^* of each estimator is inversely proportional to its own variance.

Model averaging approaches

- In model selection the winner-takes-all approach is intuitively the approach which should work the best.
- However, the final model can be improved not by choosing the one that performs apparently best but by creating a model whose output is the combination of the output of the models.
- The reason is that every hypothesis $f(\cdot)$ is only an estimate of the real target and, like any estimate, is affected by a bias and a variance term.
- A variance reduction can be obtained by simply combining uncorrelated estimators. This simple idea underlies some of the most effective techniques recently proposed in machine learning.

Stacked regression

- Suppose we have m distinct predictors $f_j(\cdot)$, for $j = 1 \ldots, m$ obtained from a given training set D.
- The idea of averaging models is to design an average estimator $\sum_{j=1}^{m} \beta_j f_j(\cdot)$
- Once computed the least-squares solution $\hat{\beta}$, the combined estimator is

$$\hat{f}(x) = \sum_{j=1}^{m} \hat{\beta}_j f_j(x)$$

Stacked regression

- The f_js are correlated because all of them are estimated on the same training set D.
- Stacked regression is an idea for combining estimators without suffering of the correlation problem. It consists in estimating the $\hat{\beta}_j$s by solving the following optimization task,

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{m} \beta_j f_j^{(-i)}(x_i) \right)^2 \quad \text{s.t.} \quad \beta_j \geq 0, \forall j.$$

- $f_j^{(-i)}(x_i)$ is the leave-one-out estimate of the jth model, i.e. the predicted outcome in x_i of the jth model trained on D with the ith sample (x_i, y_i) set aside.
- Avoids giving unfairly high weight to models with higher complexity.

Bagging

- A learning algorithm is informally called **unstable** if small changes in the training data lead to significantly different models and relatively large changes in accuracy.
- Unstable learners can have low bias but have typically high variance. Unstable methods can have their accuracy improved by **perturbing** (i.e. generating multiple versions of the predictor by perturbing the training set or learning method) and **combining**.
- **Combining multiple estimator is a variance reduction technique**. Bagging aims to improve accuracy for unstable learners by averaging over such discontinuities.

Bagging

- The idea of bagging or **bootstrap aggregating** is to imitate the stochastic process underlying the realization of D
- A set of m repeated bootstrap samples are taken from D. A model is built for each bootstrap sample and a final predictor is built by aggregating the m models
- In the regression case the bagging predictor is the average, in classification it is the majority vote.

Random Forest

- **Random forests** differ from bagging in only one way: RF use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging".
- The reason is to reduce the correlation of the trees in ordinary bootstrap samples.
- Typically, for a classification problem with p features, $\sqrt{p}$ (rounded down) features are used in each split.
- The **out-of-bag** (oob) error for each data point is recorded and averaged over the forest.

Feature importance and dissimilarity measure

- Random forests can be used to rank the **importance of variables** in a regression or classification problem in a natural way. To measure the importance of the j-th feature after training, the values of the j-th feature are permuted among the training data and the out-of-bag error is again computed on this perturbed data set.
- Random forests naturally lead to a **dissimilarity measure** between the (labeled) observations and also between unlabeled data: the idea is to construct a random forest predictor that distinguishes the "observed" data from (suitably) generated synthetic data.
- For missing values, novelty detection, variable interaction etc. Consult www.stat.berkeley.edu/$\tilde{}$breiman/RandomForests/cc_home.htm
- Variant: Extremely randomized trees. The top-down splitting in the tree learner is randomized further. Instead of computing the locally optimal feature/split combination, a random feature + value is selected for the split.

Boosting

- **Boosting** is one of the most powerful learning ideas introduced in the last ten years.
- Boosting is a general method which attempts to boost the accuracy of any given learning algorithm. It was originally designed for classification problems but it can profitably be extended to regression as well.
- Boosting encompasses a family of methods. The focus of boosting methods is to produce a series of "weak" learners in order to produce a powerful combination.
- A **weak learner** is a learner that has accuracy only slightly better than chance.

Boosting

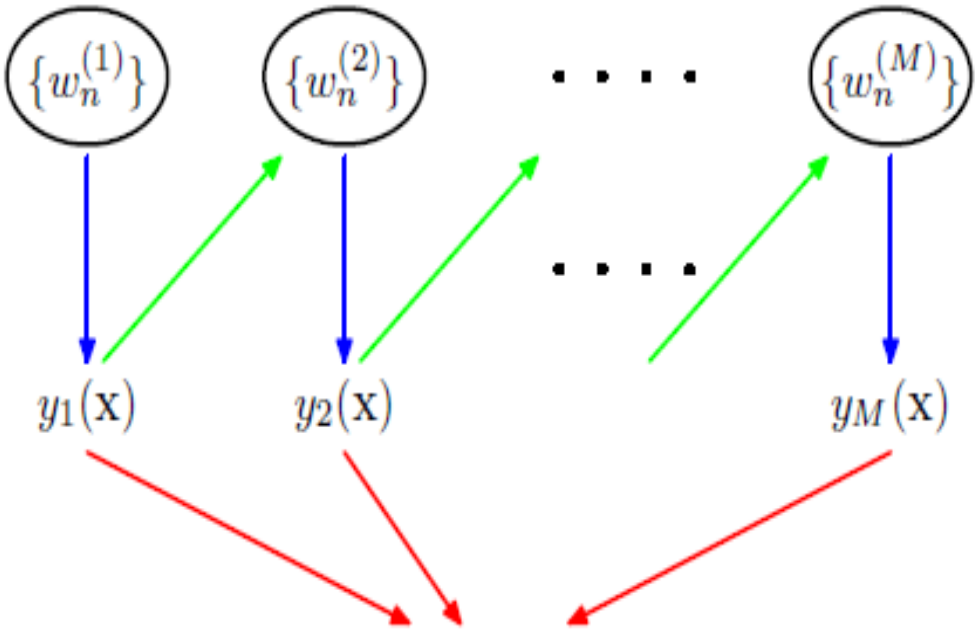
- Examples that are incorrectly predicted by previous classifiers in the series are chosen more often than examples that were correctly predicted.
- Thus Boosting attempts to produce new classifiers that are better able to predict examples for which the current ensemble's performance is poor.
- Unlike Bagging, the resampling of the training set is dependent on the performance of the earlier classifiers.
- The two most important types of boosting algorithms are the Ada Boost (Adaptive Boosting) algorithm (Freund, Schapire, 1997) and the Arcing algorithm (Breiman, 1996).

Ada Boost

- Consider a binary classification problem where the output take values in $\{-1, 1\}$.
- A weak classifier is one whose misclassification error rate is only slightly better than random guessing.
- The purpose of boosting is to sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of classifiers $f_j(\cdot)$.
- The predictions of the m weak classifiers are then combined through a weighted majority vote to produce he final prediction

$$\hat{f}(x) = \text{sign}\left(\sum_{j=1}^{m} \alpha_j f_j(x)\right)$$

Ada Boost



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Ada Boost

- The weights α_j of the different classifiers are computed by the algorithm. The idea is to give higher influence to the more accurate classifiers in the sequence.
- At each step, the boosting algorithm samples N times from a distribution that puts a weight w_i on each sample (x_i, y_i) of D.
- Initially the weights are all set to $w_i = 1/N$. Then, the weights are individually modified and the classification algorithm is re-applied to the resampled training set.

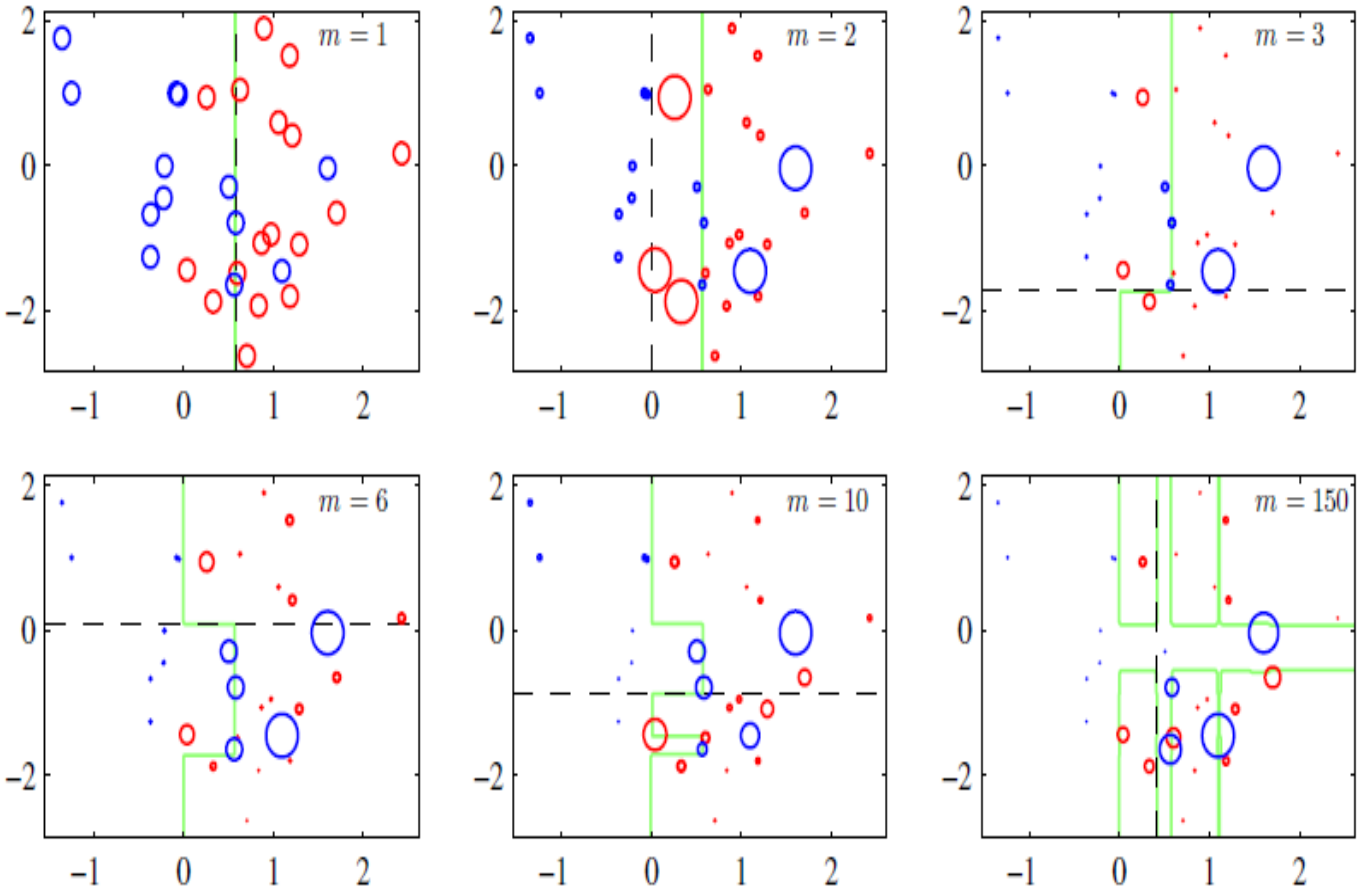
Ada Boost

- Initialize the weights to $w_i = 1/N$
- For $j = 1$ to m,
  - Fit a classifier $f_j(\cdot)$ on D using weights w_i and compute the misclassification error on the training set

$$\widehat{\epsilon}_j = \frac{\sum_{i=1}^{N} w_i \mathbb{I}[y_i \neq f_j(x_i)]}{\sum_{i=1}^{N} w_i}$$

  - Let $\alpha_j = \log((1 - \widehat{\epsilon}_j)/\widehat{\epsilon}_j)$
  - Set $w_i \leftarrow w_i \begin{cases} \exp[-\alpha_j] & \text{if correctly classified} \\ \exp[\alpha_j] & \text{if incorrectly classified} \end{cases}$
  - Normalize the weights.
- Output $\hat{f}(x) = \text{sign}\left(\sum_{j=1}^{m} \alpha_j f_j(x)\right)$.

Ada Boost



Ada Boost

- Boosting has its roots in a theoretical framework for studying machine learning called the PAC learning model.
- The empirical error of the final hypothesis f_{boo} is at most

$$\prod_{j=1}^{m} \left[2\sqrt{\widehat{\epsilon}_j \cdot (1 - \widehat{\epsilon}_j)} \right]$$

- Boosting is simple and easy to program. Moreover, it has few parameters (e.g. max number of classifiers) to tune.
- Instead of trying to design a learning algorithm which should be accurate over the entire space, one can instead focus on finding weak algorithms that only need to be better than random.
- A nice property of Ada Boost is its ability to identify outliers (hard samples).

Gradient boosting

- Like other boosting methods, **gradient boosting** combines weak "learners" into a single strong learner in an iterative fashion
- The gradient boosting method assumes a real-valued y and seeks an approximation $\hat{f}(x)$ in the form of a weighted sum of functions

$$\hat{f}(x) = \sum_{j=1}^{m} \alpha_j f_j(x)$$

- The idea is to apply a steepest descent step to this minimization problem with the following equations

$$f_m(x) = f_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{f_{m-1}} L(y, f_{m-1}(x))$$

where the derivatives are taken with respect to the functions f_i

Gradient boosting

Given a training set $\{(x_i, y_i)\}_{i=1}^n$ and a differentiable loss function $L(y, f(x))$, the generic gradient boosting method is:

1. Initialize model with a constant value $f_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$.
2. For $m = 1$ to M:
   1. Compute so-called pseudo-residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)} \qquad \forall i = 1, \ldots, n.$$

   2. Fit a base learner $h_m(x)$ to pseudo-residuals $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute γ_m by solving the 1D optimization problem:

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^n L\left(y_i, f_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model $f_m(x) = f_{m-1}(x) + \gamma_m h_m(x)$.
3. Output: $f_M(x)$.

Gradient boosting

- Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners.
- J, the number of terminal nodes in trees, can be adjusted for a data set at hand. It controls the level of interaction between variables in the model. With $J = 2$ (decision stumps), no interaction between variables is allowed. Typically $4 \leq J \leq 8$ work well.
- *XGBoost* is an open-source software library which provides the gradient boosting framework. It now has integrations with scikit-learn for Python users and supports the distributed processing frameworks Apache Hadoop, Apache Spark, and Apache Flink.
- *XGBoost* became popular among the *Kaggle* community where it has been used for a large number of competitions.

Arcing

- **Adaptive Resampling and Combining** (ARCing) was proposed as a modification of the original Ada Boost algorithms by Breiman.
- It is based on the idea that the success of boosting is related to the adaptive resampling property where increasing weight is placed on those samples more frequently misclassified.
- The complex updating equations of Ada Boost are replaced by much simpler formulations. The final classifier is obtained by unweighted voting

Arcing

- Initialize the weights to $w_i = 1/N$
- For $j = 1$ to m,
  - Fit a classifier $f_j(\cdot)$ to the training data obtained by resampling using weights w_i
  - Compute the e_i the number of misclassifications of the ith sample by the j classifiers $f_1 \ldots, f_j$
  - The updated weights are defined by

$$w_i = \frac{1 + e_i^4}{\sum_{i=1}^{N}(1 + e_i^4)}$$

  - The output is obtained by unweighted voting of the m classifiers h_j,

$$f_{\text{arc}}(x) = \text{sign}\left(\sum_{j=1}^{m} f_j(x)\right)$$